Diss. ETH No. 20371

Balanced Partitioning of Grids and Related Graphs

A Theoretical Study of Data Distribution in Parallel Finite Element Model Simulations

A dissertation submitted to

ETH Zurich

for the degree of Doctor of Sciences

presented by

Andreas Emil Feldmann
Diplom Informatiker, RWTH Aachen
born April 19, 1982,
citizen of Germany and Sweden

accepted on the recommendation of

Prof. Dr. Peter Widmayer, ETH Zurich
examiner

Prof. Dr. Christos H. Papadimitriou, UC Berkeley
co-examiner

2012

Abstract

This thesis considers the k-`BALANCED PARTITIONING` problem, which is defined as follows. Find the minimum number of edges in a graph that, when cut, partition the vertices into k (almost) equally sized sets. Amongst others, the problem derives its importance from the need to distribute data within a parallel-computing architecture. In this setting we are particularly interested in 2D finite element model (FEM) simulations. We therefore model the input as a regular quadrilateral tiling of the plane. More precisely, we focus on *solid grid graphs*. These are finite connected subgraphs of the infinite 2D grid without holes. However we also consider other graph classes. In particular, trees often give surprising conclusions to the problem on grid graphs.

For the special case when $k = 2$ (the `BISECTION` problem) we show that an optimal solution can be found in $\mathcal{O}(n^4)$ time for solid grid graphs. However the resulting runtime is unsatisfactory for practical purposes. Therefore we also show that near-optimal solutions can be found in $\mathcal{O}(n^{1.5})$ time. This is achieved by analysing the cut shapes in solid grid graphs. We prove that simple shapes corresponding to straight lines and right-angled corners suffice in order to find near-optimal cuts.

We are able to further harness these structural insights for the general case when k takes arbitrary values. This results in a fast bicriteria approximation algorithm which runs in $\mathcal{O}(n^{1.5} \log k)$ time. The number of edges it cuts is at most $\mathcal{O}(\log k)$ times the optimum and the sets are at most a factor of 2 from equal-sized. For practical purposes however, the latter deviation is unattractive. In particular in parallel-computing it means a slowdown by a factor of 2. Therefore we also consider algorithms that compute sets that are arbitrarily close to equal-sized. For trees we provide a PTAS. Furthermore, we provide a bicriteria approximation algorithm for general graphs. Here the number of edges cut is at most $\mathcal{O}(\log n)$ times the optimum. This result is obtained by harnessing results on hierarchical decompositions of graphs into trees, together with our PTAS.

Next we show that when k can be arbitrary, considering bicriteria approximations, as above, is necessary. For this we provide the following hardness results. When equal-sized sets are desired for solid grid graphs, it is NP-hard to approximate the number of cut edges within n^c, for any constant $c < 1/2$. For trees we can even prove this for any constant $c < 1$. We set up a general reduction framework in order to generate these results. The framework identifies some sufficient conditions for the considered graph class which make the problem hard. For trees the conditions are met by relying on high vertex degrees. We therefore also consider constant degree trees. For these the problem is APX-hard. This means that for trees the complexity grows with the maximum degree.

Note that one of the provided bicriteria approximation algorithms for arbitrary k is fast but yields an unsatisfactory set-size ratio. The other algorithm is slow but achieves high-quality set sizes. This is because its runtime increases exponentially when the limit on the set-sizes becomes more stringent. We justify the achieved tradeoff by showing that it is necessary. For both grids and trees we prove that, unless P=NP, no *fully* polynomial time algorithms exist that compute sets which are arbitrarily close to equal-sized. This is true even if the number of cut edges is allowed to deviate further from the optimum the more stringent the limit on the set sizes.

Zusammenfassung

Wir untersuchen das k-`BALANCIERTE PARTITIONIERUNGS` Problem, in dem eine kleinste Menge von Kanten eines Graphen gefunden werden soll, die, wenn sie geschnitten werden, die Knoten in k (fast) gleich grosse Mengen partitionieren. Dieses Problem leitet seine Bedeutung u. a. von der Notwendigkeit her, Daten auf einem Parallelrechner zu verteilen. Da uns insbesondere Simulationen von zweidimensionalen Finite-Elemente-Methoden (FEMs) interessieren, modellieren wir die Eingabe als gleichmässige viereckige Parkettierung der Ebene. Um genauer zu sein, konzentrieren wir uns auf *solide Gittergraphen*, die endliche verbundene Teilgraphen des unendlichen zweidimensionalen Gitters ohne Löcher sind. Allerdings betrachten wir auch andere Graphen. Insbesondere liefern Bäume überraschende Einsichten in das Problem auf Gittergraphen.

Im speziellen Fall wenn $k = 2$ (das `HALBIERUNGS` Problem) zeigen wir, dass eine optimale Lösung in $\mathcal{O}(n^4)$ Zeit für solide Gittergraphen gefunden werden kann. Da allerdings die Laufzeit unzufriedenstellend für praktische Anwendungen ist, zeigen wir auch, dass nahezu optimale Lösungen in $\mathcal{O}(n^{1.5})$ Zeit gefunden werden können. Dies wird durch eine Analyse der Schnittformen in soliden Gittergraphen erreicht. Wir beweisen, dass einfache Schnittformen wie gerade Linien oder rechte Winkel ausreichen, um annähernd optimale Schnitte zu finden.

Wir nutzen diese strukturellen Einsichten weiter aus, indem wir für den Fall wenn k beliebige Werte annehmen kann einen schnellen Approximationsalgorithmus über zwei Kriterien aufzeigen. Dieser läuft in $\mathcal{O}(n^{1.5} \log k)$ Zeit. Die Anzahl geschnittene Kanten ist höchstens $\mathcal{O}(\log k)$ Mal das Optimum, während die Knotenmengen höchstens um einen Faktor von Zwei von gleichen Grössen abweichen. Für praktische Anwendungen kann die letztere Abweichung allerdings nachteilig sein. Insbesondere bedeutet es eine Verlangsamung um einen Faktor von Zwei für Parallelrechner. Daher erwägen wir auch Algorithmen, die Knotenmengen berechnen, deren Grössen beliebig nah an die gewünschte Zielgrösse herankommen. Für Bäume entwickeln wir ein Polynomialzeitapproximationsschema. Weiterhin zeigen

wir einen Approximationsalgorithmus über zwei Kriterien für generelle Graphen auf. Dieser schneidet höchstens $\mathcal{O}(\log n)$ Mal so viele Kanten wie das Optimum. Dieses Resultat erhalten wir indem wir Ergebnisse über hierarchische Zerlegungen von Graphen in Bäume zusammen mit unserem Polynomialzeitapproximationsschema nutzen.

Als nächstes zeigen wir, dass wenn k beliebig sein kann das Approximieren über zwei Kriterien, wie oben, notwendig ist. Dafür stellen wir die folgenden Härtebeweise auf. Für gleich grosse Mengen ist es NP-hart die Anzahl der geschnittenen Kanten um einen Faktor n^c, für jede Konstante $c < 1/2$, zu approximieren, wenn solide Gittergraphen erwogen werden. Für Bäume können wir dies sogar für jede Konstante $c < 1$ zeigen. Wir richten ein generelles Reduktionsrahmenwerk ein, mit dem diese Resultate generiert werden können. Das Rahmenwerk identifiziert einige hinreichende Bedingungen für die erwogene Graphklasse, die das Problem hart machen. Für Bäume werden die Bedingungen erfüllt, indem hohe Knotengrade genutzt werden. Daher betrachten wir auch Bäume mit konstanten Graden. Wir zeigen, dass das Problem in diesem Fall APX-hart ist. Daher steigt für Bäume die Komplexität mit dem maximalen Grad.

Es ist auffallend, dass einer der aufgezeigten Approximationsalgorithmen für beliebige k schnell ist, aber keine zufriedenstellende Mengengrössen produziert, während der andere langsam ist, aber hochwertige Grössen berechnen kann. Dies gilt, da die Laufzeit des letzteren Verfahrens exponentiell wächst, wenn die Begrenzung der Grössen strikter wird. Wir rechtfertigen den erlangten Kompromiss, indem wir zeigen, dass er notwendig ist. Sowohl für Gitter als auch für Bäume beweisen wir, dass kein Algorithmus existiert, dessen Laufzeit *voll* polynomiell ist, und Mengen produziert, die beliebig nah an die gewünschte Zielgrösse herankommen. Dies ist sogar dann wahr, wenn die Anzahl der geschnittenen Kanten weiter vom Optimum abweichen darf, je strikter die Begrenzung der Mengengrössen wird.

Acknowledgements

To accomplish a work such as the present one, you need to stand on the shoulders of giants. These giants not only include all the intellectual forerunners throughout history on whose research I base my own, but also all the people who carried me from the cradle to where I am now. They are my family, friends, co-workers, and all those thousands of people who provided the infrastructure that made it possible for me to carry out this work in the first place. Instead of thanking the whole world at this point, I would like to only name those that contributed most directly to the writing of this thesis. All you others out there, you know who you are!

First and foremost I would like to thank Peter Widmayer for teaching me how to conduct research, by means of his relaxed laissez-fair attitude. My gratitude also goes to Christos Papadimitriou for agreeing to be my co-examiner, and for the guidance and support he gave me during our conversations. Additionally I thank Peter Arbenz for introducing me to the subject of this thesis.

Without the help of Luca Foschini my work would have been only half as good. I thank him for being an excellent co-worker and a friend. May the surf always be good for you. Apart from being a friend, Yann Disser also spent some of his precious Apple computing time for this thesis (Figure 4.1). Shantanu Das had the patience to read and correct some of my attempts at a proof.

The following people kindly read drafts of my thesis, and corrected me vigorously: Yann Disser, Annette Feldmann-Hennes, Niall Finn, Luca Foschini, Gero Greiner, Riko Jacob, Matúš Mihalák, Marcel Schöngens, and Anna Zych.

Last but not least I thank Patrick Timmann for all the punkrock and beer. “Arbeit ist scheiße; Scheißen ist Arbeit!”

Contents

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

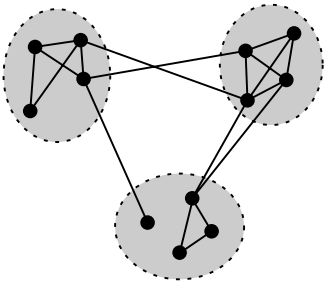


Figure 1.1: *A partition of a graph into $k = 3$ parts (indicated by grey circles) with a cut size of 5. Note that a part does not have to be a connected component.*

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

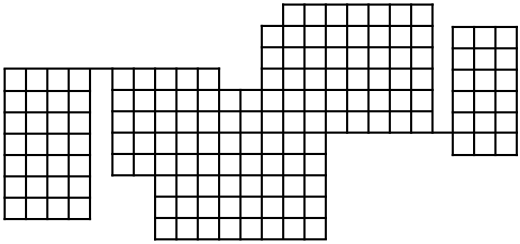


Figure 1.2: *A solid grid graph.*

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.

Chapter 2

Optimal Bisections

In this chapter we consider the problem of cutting a solid grid graph[1] into two equally sized parts and using as few edges as possible to do so. This is a special case of the k-BALANCED PARTITIONING[2] problem in which $k = 2$, and is commonly referred to as the BISECTION problem. In the following we present an algorithm that optimally solves the BISECTION problem on solid grid graphs in $\mathcal{O}(n^4)$ time. Computing approximate solutions is considered in the next chapter.

The results in this chapter were obtained in collaboration with Peter Widmayer and were published as an extended abstract [25].

2.1 Focusing on Segments

The BISECTION problem, in which a graph needs to be cut into two equal-sized parts, has its own raison d'être. One particular motivation for this problem stems from divide-and-conquer algorithms. On an abstract level the graph then represents a problem instance and the interdependencies among its elements.

[1] Definition 1.2 page 4
[2] Definition 1.1 page 3

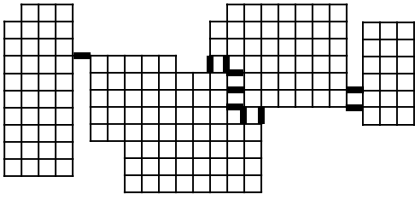


Figure 2.1: *An optimal bisection in a solid grid graph. The cut edges are indicated by being thick. Note that the parts of the bisection are not connected.*

It needs to be divided into two equal-sized sub-instances which are recursively solved. The two solutions are then put together to form a solution to the overall instance. Typically this can be done faster the fewer dependencies exist between the two sub-instances. For example the `MINIMUM FEEDBACK ARC SET` problem can approximately be solved in polynomial time using this technique [64]. This problem frequently appears in applications such as analysing the control structure of a computer program, and code optimisation [59].

In the following sections we show how to compute an optimal solution to the `BISECTION` problem for solid grid graphs (Figure 2.1). Any partition of the vertex set of a graph into two sets of size at most $\lceil n/2 \rceil$ each is called a *bisection*. The cut size of an optimal bisection is called the *bisection width* of the given graph. Formally the problem is defined as follows.

Definition 2.1 (`BISECTION`)**.** Given a graph $G = (V, E)$ find a bisection that minimises the cut size.

The presented algorithm to solve this problem uses a dynamic programming scheme which is based on recursively building the cut using basic building blocks called *segments*. To define them it is convenient to consider the *dual graph* of a grid graph. For any planar graph G the dual (multi-)graph D contains the faces of an embedding of G as vertices. There is an edge in D between any two faces that share an edge of G. We say that these edges in G and D *correspond* to one another.

Definition 2.2 (segment)**.** Given a planar graph $G = (V, E)$ and its dual graph D, a set of edges $s \subseteq E$ of G is called a *segment* if the set of corresponding edges in D form a simple cycle.

Instead on concentrating on the vertex sets of the desired bisection we will focus on the segments that are needed to cut them out. Accordingly we define a *cut* as a set of segments. Our algorithm goes through each segment in a solid grid graph and recursively compute a partial solution for it. The main reason why optimal bisections can be computed in polynomial time for solid grid graphs is that there are only a polynomial number of segments that need to be considered. In order to show our claimed runtime of $\mathcal{O}(n^4)$ it is therefore necessary to thoroughly analyse the structure of the needed segments.

2.1.1 An Overview of the Used Techniques

Our approach is based on two key findings. The first one limits the shapes of the segments needed to bisect a solid grid, and it bounds the number of cut edges in an optimum cut. The second one uses these limits in a dynamic program.

We recall from the work by Papadimitriou and Sideri [54] that it is enough for an optimum cut to limit the segments to the shape of a straight line, a corner, a stair, a clamp, or a square (cf. Figure 2.2), with one small variation: for the sake of being able to cut off exactly the desired number of vertices, a side-step by one grid cell can be present that we call a *break* and define precisely later. Furthermore, we show that a single stair, clamp, or square segment (with or without break) is enough in an optimum cut. That is, all others can be straight and corner segments. In addition, we can bound the number of edges of any segment in an optimum cut to $\mathcal{O}(\sqrt{n})$, by recalling from the results of Diks *et al.* [18] that the bisection width of a solid grid graph on n vertices is $\mathcal{O}(\sqrt{n})$.

We make use of these limitations on segments by explicitly considering all possible stairs, clamps, and squares, without and

with a break. For each of both parts into which such a segment cuts the grid, we only consider straight and corner segments that cut away exactly the desired number of vertices. We are able to compute the optimal way to cut out any desired number of vertices in each part using only straight and corner segments in a dynamic program. This is done by splitting a part into smaller pieces and recursively computing a solution for these. A part has to be split several times in order to guarantee that the optimal solution is found. We will define a set of splits that each uses a subset of the considered segments to do so. We will show that the number of necessary splits is small enough to guarantee the claimed runtime.

The efficiency of our approach rests on the fact that there are only $\mathcal{O}(n^2)$ segments to be considered, since each segment is defined by three parameters: first, one of the corners in its shape (at one of at most n positions); second, the distance to a (suitable) neighbouring corner of its shape (at most $\mathcal{O}(\sqrt{n})$); and third, the potential position of a break (at most $\mathcal{O}(\sqrt{n})$ possible ones). We will show that only an additional multiplicative term of $\mathcal{O}(n^2)$ is needed to compute the optimal bisection. This proves the claimed runtime of $\mathcal{O}(n^4)$. Hence we improve on the formerly fastest known algorithm for solid grid graphs by Papadimitriou and Sideri [54] which runs in $\mathcal{O}(n^5)$ time.

2.1.2 Related Work

Since this chapter is only concerned with solving the `BISECTION` problem optimally, we defer the survey of related approximation results to Section 3.1.1. For general graphs the `BISECTION` problem is NP-hard [31]. This is true even if the graph has maximum degree 3 [47], or it is regular [9]. Determining the complexity of the `BISECTION` problem on planar graphs remains a challenging open problem. If however weights are allowed on the vertices it is possible to show that it is weakly NP-hard for these graphs [55]. Also the complexity on grid graphs with holes is not known. However Papadimitriou and Sideri [54] gave a reduction from planar graphs to grid graphs with holes. They introduce a huge (but polynomially sized) square shaped

solid grid for each vertex in the embedding of the planar graph. Each edge of the latter is substituted by a path between the corresponding square grids. The resulting grid graph has the same bisection width as the original planar graph.

For special graph classes polynomial time algorithms are known. For instance, MacGregor [47] gave an $\mathcal{O}(n^3)$ time algorithm for trees. This algorithm is of central importance to the algorithm for solid grid graphs presented in this chapter. It is a dynamic program that recursively considers each edge in a tree in a bottom-up fashion. More precisely, it computes the optimal way to cut out m vertices from the currently considered subtree rooted at an edge e, for each m. In order to do so it decides whether to cut e using the precomputed solutions for the subtrees rooted at the edges immediately below e. These solutions first have to be combined in order to find the optimum way to cut out m vertices from all these subtrees. This combination technique, known as computing the *min-convolution*, is also used in our algorithm presented in this chapter. Computing the min-convolution can be done in linear time. Since for each edge in the tree an optimal solution has to be computed for every possible value of m, the runtime of $\mathcal{O}(n^3)$ follows. Using a more sophisticated amortised runtime analysis it is possible to show that MacGregor's algorithm actually runs in time $\mathcal{O}(n^2)$. This was noted in [32] and a corresponding runtime analysis can be found in [28] for a related problem.

The algorithm presented in this chapter improves on the formerly fastest known algorithm for solid grid graphs by Papadimitriou and Sideri [54] which runs in $\mathcal{O}(n^5)$ time. Their algorithm is based on the intuition that a solid grid resembles a discretised simple polygon. Accordingly their work was extended [41] to compute (approximate) bisections in simple polygons, for which it was also shown that the problem is NP-hard. For solid grid graphs, it was shown [54] that in an optimal solution all segments correspond to simple cycles containing the exterior face in the dual graph. Hence all segments end at the boundary of the given solid grid graph. The idea is then to consider two edges at the boundary of the grid which split the boundary into two continuous pieces. For each such piece a solution is computed

that contains only segments that end at this piece. This is done recursively using the precomputed solutions for those pieces of the boundary that are contained in the considered one. In particular the considered piece of the boundary is split into two by each edge lying on it. Hence the resulting dynamic programming scheme needs to consider all $\mathcal{O}(n^2)$ pairs of edges to define a piece of the boundary. Additionally it needs to split such a piece using each of the at most $\mathcal{O}(n)$ edges in it. Since the algorithm needs $\mathcal{O}(n^2)$ time to compute the optimal m-cuts for each such triple of edges, the overall runtime is $\mathcal{O}(n^5)$.

Other graph classes for which optimal solutions can be computed in polynomial time include hypercubes [52], ordinary two and three dimensional meshes [61], and bounded tree-width graphs [66]. For a survey see [15]. In the PRAM model Goldberg and Miller [32] show that the `BISECTION` problem on trees can be solved in $\mathcal{O}(\log^2 n \log\log n)$ time on $\mathcal{O}(n^2)$ processors.

2.2 Properties of Optimal m-Cuts

For our dynamic program we need to generalise the `BISECTION` problem to cutting out any number m of vertices.

Definition 2.3 (m-cut)**.** An *m-cut* is a set of segments that, when removed from the graph, leaves a spanning subgraph that contains a set of connected components including exactly m vertices. We say that an m-cut S *cuts out* the m vertices of these connected components. The *A-part* of an m-cut S is the set of vertices of size m cut out by S. Accordingly the *B-part* contains the other $n - m$ vertices. Given an m-cut S we call the number of edges $\sum_{s \in S} |s|$ in S its *cut size*, and we call an m-cut that minimises the cut size over all m-cuts *optimal*.

Notice that some edges may be counted several times in the sum measuring the cut size. However, edges that appear more than once in different segments can be removed from an m-cut. This is why this generalisation does not change the optimal solution.

In order to prove our claimed results we need to analyse the types of segments that may occur in an optimal m-cut. We first recall from [54] that these only include so called straight, corner, stair, clamp, and square segments. Thereafter we will prove that at most one of the segments in an optimal m-cut is not a straight or corner segment. We begin by formally defining the above types of segments (Figure 2.2) with the help of the dual (multi-)graph D of a solid grid graph G. In the following any face refers to a face of G, i.e. a vertex of D. Let s be a segment in G such that the simple cycle p corresponding to s in the dual D includes the exterior face. An interior face f in D lying on p is called a *bend* of s if f touches two edges $e_1, e_2 \in s$ such that e_1 and e_2 share a vertex. We say that the bend f *points* in two *directions*: the directions are *up* and *right* if e_1 and e_2 lie above and to the right of f, and analogously they can be *down* or *left* if the edges lie appropriately. Two bends of s are said to *point in opposing directions* if they do not share any direction in which they point. If they share at least one direction they are said to *point in a common direction*. A *break* of s is an edge $e \in s$ such that e touches two bends of s that point in opposing directions. Let q be a sub-path of p such that q starts and ends in two faces f_1, f_2 of D and f_1 and f_2 are bends of s or equal the exterior face f_∞. The subset b of s corresponding to q is called a *bar* of s if no face on q between f_1 and f_2 is a bend of s or equal to f_∞. The bar b is said to *end at* the two faces f_1 and f_2. The subset b is called a *broken bar* of s if b can be partitioned into three bars of s and the one ending neither at f_1 nor at f_2, i.e. the middle one, consists of a break of s. Also the broken bar b is said to *end at* the two faces f_1 and f_2. Two bends of s are called *consecutive* if there is a bar of s ending at them.

Definition 2.4 (straight, corner, stair, clamp, square segment)**.** A segment s is called

- a *straight* segment if it has no bend.
- a *corner* segment if it has exactly one bend.
- a *stair* segment if any consecutive bends of s point in opposing directions. Additionally, if s has no break then it has exactly two bends. Otherwise it contains a broken

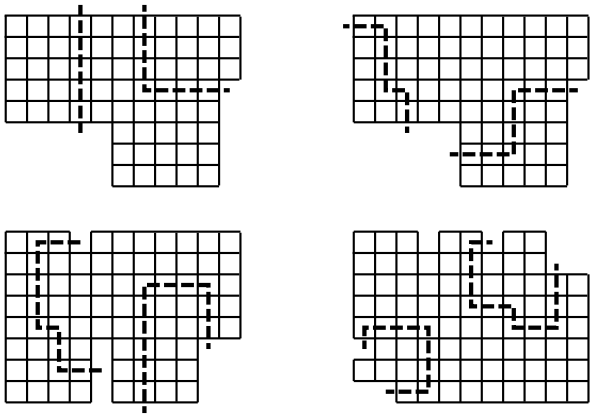


Figure 2.2: *The types of segments occurring in optimal m-cuts. In this order from left to right: straight and corner segments, stair segments with and without breaks in the top row. Clamp segments with and without breaks, and square segments without and with breaks in the bottom row.*

bar b such that $s \setminus b$ constitutes at most two bars each of which ends at f_∞.

- a *clamp* segment if there are two bends f_1, f_2 of s pointing in a common direction. Additionally s can be partitioned into (1) a bar or broken bar b ending at f_1 and f_2, and (2) two bars that both end at f_∞.

- a *square* segment if there are three bends f_1, f_2, f_3 of s such that s can be partitioned into (1) a bar or broken bar b ending at f_1 and f_2, (2) a bar b' ending at f_2 and f_3, and (3) two bars ending at f_∞ and f_1 or f_3, respectively. Additionally each of the pairs f_1, f_2 and f_2, f_3 point in a common direction, and if b is a broken bar then f_1 and its consecutive bend of s touching the break, point in a common direction. Moreover $|b'| \leq l \leq |b'| + 1$, where l is the length of b without counting its breaks. That is, $l = |b|$ if b is a bar, and $l = |b| - 1$ if b is a broken bar.

In the latter three cases, if b is a broken bar we refer to its break

as *the break of* s. For a corner, clamp, or square segment s let $v \in V$ be the vertex of the grid graph that is shared by the edges touching the bend f_1 of s. The part cut out by s including v is referred to as *convex*, and the other part as *concave*.

The next lemma states that there exists an optimal m-cut containing only the types of segments in the above definition. It follows from observations given in [54].

Lemma 2.5 (follows from [54])**.** *There is an optimal m-cut containing only straight, corner, stair, clamp, and square segments.*

Proof sketch. In their paper, Papadimitriou and Sideri identify the same types of segments as given in Definition 2.4 for an optimal m-cut. However they also name segments which have the shape of a *staircase* as possible candidates [54, Lemma 4]. These are segments that are monotone in x- and y-direction and hence represent a generalisation of stair segments, as defined above. Additionally the authors present a procedure [54, Lemma 3] with which it is possible to convert a staircase shaped segment into a stair segment. It considers the symmetric difference between the cut out parts of the two segments. This difference is decreased by finding two vertices in each of the two parts cut out by the staircase shaped segment, and swapping them between the parts. Moreover, this can be done so that the resulting parts are again cut out by a staircase shaped segment of the same length. By repeating this procedure the m-cut is transformed into one that contains only segment types as listed in Definition 2.4. □

Furthermore at most one segment in an optimal m-cut is not a straight or corner segment. We prove this by shifting pieces of cut out areas from one segment to another. This is done so that the overall cut out area stays the same while the cut size does not increase.

Theorem 2.6. *There is an optimal m-cut that contains only straight and corner segments except at most one which is either a stair, clamp, or square segment.*

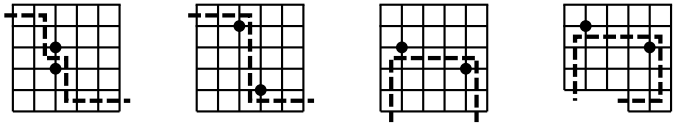


Figure 2.3: *The vertices v_A and v_B (large dots) of a segment with break, and a stair, clamp, and square segment without break, respectively.*

Proof. According to Lemma 2.5 we can assume that an optimal m-cut S contains only straight, corner, stair, clamp, and square segments. Let A and B be the parts cut out by S. It is easy to see that in an optimal m-cut there are no two segments sharing an edge, since such an edge could be removed from both segments to yield an m-cut of smaller cut size. Hence all vertices incident to an edge in some segment $s \in S$ and belonging to one of the cut out parts of s, also belong to either one of A or B. We assume that there are at least two segments in S that are not straight or corner segments and show how to convert S into another optimal m-cut containing at most one of these types of segments. For this we will identify two vertices $v_A \in A$ and $v_B \in B$ for any stair, clamp, and square segment s, and are incident to some edge from s each. Such a vertex v_P, where $P \in \{A, B\}$, can be removed from P and added to the other part $\overline{P} \in \{A, B\} \setminus \{P\}$ by only slightly changing the shape of s (Figure 2.3). More concretely, of the incident edges to v_P some belong to s and others do not. The vertex v_P will be chosen such that the segment s' including all edges from s except those incident to v_P, but additionally including those incident to v_P that are not contained in s, is a straight, corner, stair, clamp, or square segment. Hence if s is exchanged with s' the set P loses the vertex v_P in the cut. Notice that the segment s' may not exist since some of the edges incident to v_P may touch the exterior face of the grid graph. In this case the claimed segment s' is split into (at least) two segments by the boundary of the grid graph. Due to [54, Lemma 3], it is however possible to convert the resulting segments into straight, corner, stair, clamp, or square segments analogous to the way it is done for Lemma 2.5

above. These segments can then just as well be used in the following arguments and hence this special case will be ignored from now on.

Since we assume that S contains at least two segments that are not straight or corner segments, we can find another segment $t \in S$ such that we can give back the lost vertex to P. We do this by accordingly exchanging t with a segment t' such that a vertex $v_{\overline{P}}$ from the other part $\overline{P} \in \{A, B\} \setminus \{P\}$ is lost from $\overline{P}$. Hence after exchanging both s and t with s' and t' respectively, the new set of segments is again an m-cut containing only straight, corner, stair, clamp, and square segments. We need however to make sure that the cut size of the resulting m-cuts is non-increasing during these exchange steps, i.e. a step in which two segments are exchanged, in order to preserve optimality. This is not always possible for single exchange steps but we will show that there always exists a series of exchange steps for the considered segments s and t that lead to non-increasing cut sizes.

We now list the vertices v_A and v_B for each stair, clamp, and square segment. For any segment s with a break the two vertices are those incident to the break of s. If s is a stair segment without break then let f_1 and f_2 be the two bends of s. In this case one of the two vertices v_A and v_B is the one shared by the two edges from s touching the bend f_1, and the other vertex is the one shared be the two edges from s touching f_2. If s is a clamp or square segment without break then let f_1 and f_2 be the bends and b the bar of s as in Definition 2.4. In this case one of the two vertices v_A and v_B is the one incident to the edge from b touching the bend f_1 and lying in the concave part cut out by s. The other vertex is the one incident to the edge from b touching f_2 and lying in the convex part cut out by s. It is easy to check that in all of these cases each of the vertices v_A and v_B has the property that the corresponding segment s', as described above, is again a straight, corner, stair, clamp, or square segment. In particular, also for square segments the properties on the lengths of the two (broken) bars b and b' are preserved.

If a vertex v_P, where $P \in \{A, B\}$, has more incident edges that are not contained in the respective segment s than edges that are, then the corresponding segment s' contains more edges

than s and thus the cut size may increase if s is exchanged with s'. Note however that this can only happen if s is a clamp or square segment without break and v_P is contained in the concave part cut out by s. Therefore if S contains two stair segments s and t, we can use the vertex v_A for s and v_B for t to exchange s and t with s' and t' respectively without increasing the cut size. It cannot be that s' or t' shares an edge with some other segment from the resulting m-cut, since such a shared edge could be removed which would result in an m-cut of smaller cut size. Hence this exchange step can be repeated until one of the two segments is replaced with a straight or corner segment. Thus the resulting m-cut is optimal and has one stair segment less than S. Similarly if s is a clamp or square segment and t is a stair segment, we fix $P \in \{A, B\}$ to be the set for which v_P for s lies in the convex part cut out by s. We can then exchange s and t with corresponding segments s' and t' repeatedly without increasing the cut size. Clearly also in this case one of the segments will at some point be exchanged by a straight or corner segment.

If both s and t are clamp or corner segments let the respective (broken) bars referred to as b in Definition 2.4 be b_s and b_t. We assume w.l.o.g. that $|b_s| \leq |b_t|$. Fix $P \in \{A, B\}$ such that v_P for s lies in the convex part cut out by s. If $v_{\overline{P}}$ for t lies in the convex part of t then clearly we can find a sequence of exchange steps as above for which the cut size does not increase. Also in the resulting m-cut one of the segments was subsequently replaced by a straight or corner segment. However if $v_{\overline{P}}$ lies in the concave part, as noted above the cut size is increasing if t has no break. Therefore in this case we need to consider a sequence of exchange steps until s is subsequently replaced with a segment s' that does not have a break. The number of these steps is at most $|b_s|$ and the segment s' contains two edges less than s. Since $|b_s| \leq |b_t|$, there can be at most one segment subsequently replacing t that does not have a break in this sequence of steps. Hence the corresponding resulting segment t' can only have at most two edges more than t. All vertices that are exchanged between A and B during this sequence are incident to edges in s and t in the respective convex and concave cut out parts. Thus in each step the vertices v_A and v_B always exist for the corresponding segments in the exchange sequence. As above it

cannot happen that s' and t' share edges with other segments in the resulting m-cut since this would be a contradiction to optimality. Hence also for this case we can find a sequence of exchange steps for which the cut size is non-increasing and the resulting m-cut contains one clamp or square segment less than S.

The proof is concluded by noticing that we can repeat this procedure for all pairs of segments that are neither straight nor corner segments until at most one such segment remains. □

As a consequence of the above theorem the obvious way to proceed at this point would be to consider each stair, clamp, and square segment explicitly in the algorithm, as described in the introduction. However the runtime would in this case be larger than claimed since, for instance, there are $\Theta(n^3)$ stair segments in the worst case. Not all of these will appear in an optimal m-cut though, since some of them are too large. This follows from the fact that the maximum degree of a grid graph is 4, and a result by Diks *et al.* [18] who showed that the *cut width* of any planar graph of maximum degree Δ is $\mathcal{O}(\sqrt{\Delta n})$. The cut width is defined as follows. For any graph a *linear layout* is a bijection f of the vertex set V to $\{1, \ldots, n\}$. The *cut width* of f is the maximum number of edges over all $i \in \{1, \ldots n-1\}$ that connect those vertices $v \in V$ for which $f(v) \leq i$ with those for which $f(v) > i$. For a graph the cut width is the minimum cut width over all its linear layouts. Clearly the cut width of a planar graph[3] is an upper bound on the cut size of the optimal m-cut for any m. Thus the following theorem follows.

Theorem 2.7 (follows from [18])**.** *The cut size of any optimal m-cut in a grid graph is $\mathcal{O}(\sqrt{n})$.*

This means that by further restricting some of the segments to such ones that contain at most $\mathcal{O}(\sqrt{n})$ edges we are able to reduce the runtime.

[3]This can easily be generalised to non-planar graphs by defining m-cuts accordingly for that setting.

2.3 Computing Optimal m-Cuts

In this section we present an algorithm to compute optimal m-cuts in solid grid graphs. We do this by assuming that we are given a solid grid graph G and a family $\mathcal{S}$ of straight, corner, stair, clamp, and square segments in G. In order to obtain the claimed runtime, of the segments with breaks we include only those having a length of $\mathcal{O}(\sqrt{n})$ in $\mathcal{S}$. According to Lemma 2.5 and Theorem 2.7 it suffices to compute an optimal m-cut that only uses segments from $\mathcal{S}$. More formally, for any family of segments $\mathcal{T}$ we say that any m-cut S is *$\mathcal{T}$-restricted* if $S \subseteq \mathcal{T}$, and our goal is to compute an optimal $\mathcal{S}$-restricted m-cut. Additionally we assume that we are given the set $\mathcal{C} \subseteq \mathcal{S}$ of straight and corner segments in G. We call an m-cut that is $\mathcal{C}$-restricted a *corner m-cut*. According to Theorem 2.6 we know that any optimal $\mathcal{S}$-restricted m-cut contains at most one segment that is not from $\mathcal{C}$.

One crucial observation needed to construct the dynamic program is that we can assume that no segments cross in the optimal m-cut. Note that a simple cycle in the dual of a planar graph corresponds to a closed curve in the embedding of the dual graph in the plane. Hence the cycle divides the plane into an interior and an exterior area. We say that a pair of cycles *cross* if the corresponding closed curve of one of them, both contains points belonging to the interior and the exterior area into which the other cycle divides the plane. Note that any pair of simple cycles that cross can be seen as a (different) pair of simple cycles

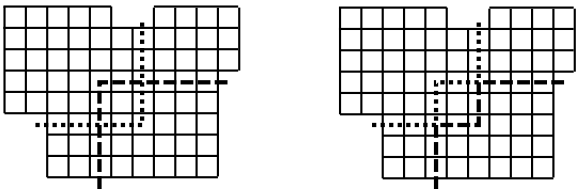


Figure 2.4: *A crossing and an equivalent non-crossing m-cut containing two segments. The segments are indicated by a dashed and a dotted line.*

that do not cross (Figure 2.4). Hence we may limit ourselves to cuts in which no segments cross and we call these *non-crossing*.

The idea behind the algorithm is to guess a stair, clamp, or square segment $s \in \mathcal{S} \setminus \mathcal{C}$ from which we know that it is contained in the optimal solution and all other segments are straight and corner segments from $\mathcal{C}$. The case when the optimum is a corner m-cut is dealt with separately. We split the graph into the two parts $A(s)$ and $B(s)$ cut out by s. If the optimum corner cuts in these two parts are known, then these can be used to compute the optimum containing s. That is, we can compute the optimal cut size $C_s(m)$ of a non-crossing m-cut that contains s and only segments from $\mathcal{C}$ otherwise. Let for a part $P \in \{A(s), B(s)\}$ the set $\mathcal{C}_P$ include every segment $t \in \mathcal{C}$ that cuts out a part $A(t) \subseteq P$. Let also $C_P(m)$ denote the optimal cut size of a $\mathcal{C}_P$-restricted m-cut such that the m vertices are cut out from P. We define the value of $C_P(m)$ to be infinite if no such cut exists. Using $C_{A(s)}(\cdot)$ and $C_{B(s)}(\cdot)$ we compute $C_s(m)$ as follows. The corresponding m-cut cuts out some number m' of the vertices from $A(s)$, and the remaining $m - m'$ from $B(s)$. The cuts in $A(s)$ and $B(s)$ must be of minimum cut size since they could otherwise be exchanged with better ones decreasing the overall cut size $C_s(m)$. Since s is included in the m-cut, the cut size of the segments cutting out vertices from $A(s)$ is either $C_{A(s)}(m')$ or $C_{A(s)}(|A(s)| - m')$. In the former case the cut size of the segments in $B(s)$ is $C_{B(s)}(|B(s)| - (m - m'))$, while in the latter it is $C_{B(s)}(m - m')$. Thus the optimal cut size is

$$C_s(m) = \min\{|s| + C_{A(s)}(|A(s)| - m') + C_{B(s)}(m - m'), \\ |s| + C_{A(s)}(m') + C_{B(s)}(|B(s)| - (m - m')) \mid \\ m' \in \{0, ..., m\}\}. \quad (2.1)$$

According to Theorems 2.6 and 2.7, taking the minimum over all $s \in \mathcal{S} \setminus \mathcal{C}$ of all computed values $C_s(m)$ correctly computes the optimal m-cut if it contains a segment from $\mathcal{S} \setminus \mathcal{C}$. To handle the case when the optimum only contains segments from $\mathcal{C}$, we define $C_P(m)$ and $\mathcal{C}_P$ accordingly for any $s \in \mathcal{C}$ and $P \in \{A(s), B(s)\}$. Notice that then $s \in \mathcal{C}_P$ since s is a segment from $\mathcal{C}$ that cuts out P. We treat this special case by also taking the cut size $C_P(m)$ of a segment $s \in \mathcal{C}$ for which $\mathcal{C}_P = \mathcal{C}$ into account in the

final step. Such a segment can easily be found by considering an arbitrary segment in $\mathcal{C}$ that cuts out a single vertex from the given grid graph. Hence also considering the corresponding value $C_P(m)$ correctly finds the optimal solution. Note that given the functions $C_P(\cdot)$, for a fixed m the algorithm takes $\mathcal{O}(\sum_{s\in\mathcal{S}\setminus\mathcal{C}} n)$ time to compute the optimum m-cut according to Equation (2.1). This is because for each segment $s \in \mathcal{S} \setminus \mathcal{C}$ it needs to consider all possible values for m'.

It remains to be shown how the optimal cut sizes $C_{A(s)}(\cdot)$ and $C_{B(s)}(\cdot)$ of the corner cuts in the parts $A(s)$ and $B(s)$ are computed for any $s \in \mathcal{S}$. The main inspiration for this part of our algorithm is taken from the corresponding algorithm for trees [47], as described in Section 2.1.2. In a tree the segments correspond to single edges and a dynamic program is used to compute an optimal m-cut bottom-up from the leaves to the root. For each edge e of the tree the algorithm computes the optimal solution for the subtree at e. It then decides whether to include e in the solution by considering the optimal cuts in the subtrees immediately below e. Combining the cuts in the subtrees in order to compute the optimum up to e is easy since they do not interfere with one another.

For our case we proceed in a similar way as for trees by computing the cut size $C_P(m)$ of an optimal $\mathcal{C}_P$-restricted m-cut for every $m \in \{0, ..., n\}$ in each part $P \in \{A(s), B(s)\}$ cut out by a segment $s \in \mathcal{S}$. We will decide whether to include s into the solution for the part P by considering the cuts computed for segments cutting out parts from P. However these solutions do interfere with one another since the parts can overlap. In order to circumvent this problem the idea is to guess where P has to be split so that each segment of the non-crossing optimum is contained in one of the resulting pieces of P. We use segments from $\mathcal{S}$ cutting out parts in P for splitting in order to be able to proceed recursively. To find the correct way to split a part we give the following definition.

Definition 2.8 (IFS)**.** Let $P \in \{A(s), B(s)\}$ be a part cut out by a segment s and let S denote a set of segments such that each $t \in S$ cuts out a part $A(t) \subset P$. The set S is called an *interference-free set (IFS) in* P if $A(t) \cap A(t') = \emptyset$ for each pair

$t \neq t'$ from S. Let $\mathcal{P}(S)$ be the set containing all parts $A(t) \subset P$ cut out by the segments $t \in S$. Let also the set $\mathcal{C}_{\mathcal{P}(S)}$ contain all segments from $\mathcal{C}_P$ that cut out a part included in $A(t)$ for some $t \in S$.

Note that s itself cannot be contained in an IFS in $P \in \{A(s), B(s)\}$. In order to find the cut size of an optimal non-crossing corner m-cut in P we will split P according to each IFS from a small predefined set of IFSs in P. We will need one such predefined set for each part cut out by a segment $s \in \mathcal{S}$ and hence call them $\mathcal{K}_{A(s)}$ and $\mathcal{K}_{B(s)}$ respectively. In the next section we will show that for each considered cut out part we can find such a set that is small enough in order to guarantee the claimed runtime. The IFSs in the set include segments from $\mathcal{S}$ and together have the property that they cover all IFSs including segments from $\mathcal{C}$ in P, in the following sense.

Definition 2.9 (IFS covering set)**.** Let $s \in \mathcal{S}$ cut out a part $P \in \{A(s), B(s)\}$. An *IFS covering set* $\mathcal{K}_P \subseteq 2^{\mathcal{S}}$ includes IFSs in P such that for any IFS $S \subseteq \mathcal{C}$ also in P, there is a set $S^* \in \mathcal{K}_P$ for which $S \subseteq \mathcal{C}_{\mathcal{P}(S^*)}$.

Fix an IFS covering set $\mathcal{K}_P$ for the part P cut out by the segment s we are considering. For any IFS S in P let $C_{\mathcal{P}(S)}(m)$ denote the optimal cut size of a $\mathcal{C}_{\mathcal{P}(S)}$-restricted m-cut such that the m vertices are cut out from P. We define the value of $C_{\mathcal{P}(S)}(m)$ to be infinite if no such cut exists. To compute the cut size of the optimal corner m-cut in P we can split P according to each IFS $S^* \in \mathcal{K}_P$ and make use of the functions $C_{\mathcal{P}(S^*)}(\cdot)$. To see this we show that any non-crossing corner m-cut T in P is $\mathcal{C}_{\mathcal{P}(S^*)}$-restricted for some $S^* \in \mathcal{K}_P$. Consider the set T' of segments containing any $t \in T \setminus \{s\}$ that cuts out a part $A(t) \subset P$ such that there is no other segment $t' \in T \setminus \{s\}$ that cuts out a superset $A(t') \subset P$ of $A(t)$. Since T is non-crossing the set T' is an IFS. Since also $T' \subseteq \mathcal{C}$, by the definition of an IFS covering set this means that $T \setminus \{s\}$ is $\mathcal{C}_{\mathcal{P}(S^*)}$-restricted for some $S^* \in \mathcal{K}_P$. Hence the optimal non-crossing corner m-cut in P is $(\mathcal{C}_{\mathcal{P}(S^*)} \cup \{s\})$-restricted for some $S^* \in \mathcal{K}_P$.

By the above observation, we need only consider the sets $S^* \in \mathcal{K}_P$ and pick the solution that has the minimum cut size

according to the functions $C_{\mathcal{P}(S^*)}(\cdot)$ to compute the optimum in P. If $s \in \mathcal{C}$ we also need to consider the case when s is included in the solution. In this case the number of cut out vertices from P is $|P| - m$. Hence for any $s \in \mathcal{S}$ and $P \in \{A(s), B(s)\}$, in case the IFS covering set $\mathcal{K}_P$ is non-empty, if $s \in \mathcal{S} \setminus \mathcal{C}$ then

$$C_P(m) = \min\{C_{\mathcal{P}(S^*)}(m) \mid S^* \in \mathcal{K}_P\}, \tag{2.2}$$

and if $s \in \mathcal{C}$ then

$$C_P(m) = \min\{C_{\mathcal{P}(S^*)}(m), |s| + C_{\mathcal{P}(S^*)}(|P| - m) \mid S^* \in \mathcal{K}_P\}. \tag{2.3}$$

In the other case when $\mathcal{K}_P$ is empty there are no segments that cut out a subset of P. Hence then $C_P(0) = 0$ and if $s \in \mathcal{C}$ also $C_P(|P|) = |s|$. All other values of $C_P(\cdot)$ are infinite. Thus computing the table containing all values of the functions $C_P(\cdot)$ takes $\mathcal{O}(n \cdot \sum_{s \in \mathcal{S}} |\mathcal{K}_{A(s)} \cup \mathcal{K}_{B(s)}|)$ steps if all values of the functions $C_{\mathcal{P}(S^*)}(\cdot)$ of corresponding IFSs S^* are given.

The last missing part of this section is to show how a function $C_{\mathcal{P}(S)}(\cdot)$ for a non-empty IFS S in a cut out part P of a segment from $\mathcal{S}$ can be computed. In order to find the cut size $C_{\mathcal{P}(S)}(m)$ of an optimal $\mathcal{C}_{\mathcal{P}(S)}$-restricted m-cut, the algorithm will combine the solutions computed for the segments in S in the same way the solutions for subtrees were combined in the algorithm for trees in [47]. The used technique is known as forming the *min-convolution*. If S contains only a single segment t then obviously $C_{\mathcal{P}(S)}(m) = C_{A(t)}(m)$, where $A(t) \in \mathcal{P}(S)$. In case S contains more than one segment, the value of $C_{\mathcal{P}(S)}(m)$ can, for any fixed $t \in S$, be recursively computed using the solutions to the IFS $S \setminus \{t\}$ and the solution for the part $A(t) \in \mathcal{P}(S)$. An optimal $\mathcal{C}_{\mathcal{P}(S)}$-restricted m-cut must cut out some number m' of the m vertices cut from $A(t)$. The remaining $m - m'$ vertices are taken from the parts in $\mathcal{P}(S \setminus \{t\})$. Thus finding the minimum cut size among all possible values of m' will find the optimal solution. Hence the following equation is correct.

$$C_{\mathcal{P}(S)}(m) = \min\{C_{A(t)}(m') + C_{\mathcal{P}(S\setminus\{t\})}(m - m') \mid A(t) \in \mathcal{P}(S) \wedge m' \in \{0, ..., m\}\}. \tag{2.4}$$

1. **for all** $s \in \mathcal{S}$ **and all** $S^* \in \bigcup_P \mathcal{K}_P$ **do:**
   - **compute** C_P using $C_{\mathcal{P}(T^*)}$, (Eqn. (2.2), (2.3)) where $P \in \{A(s), B(s)\}$ and $T^* \in \mathcal{K}_P$.
   - **compute** $C_{\mathcal{P}(S^*)}$ using C_P, (Eqn. (2.4)) where $P \in \mathcal{P}(S^*)$.
2. **for all** $s \in \mathcal{S} \setminus \mathcal{C}$ **do:**
   - **compute** C_s using $C_{A(s)}$ and $C_{B(s)}$. (Eqn. (2.1))
3. **return** $\min\{C_s(m), C_P(m) \mid s \in \mathcal{S} \setminus \mathcal{C} \ \wedge \ \mathcal{C}_P = \mathcal{C}\}$

Figure 2.5: *The overall structure of the algorithm to compute an optimal m-cut.*

To evaluate the right hand side of Equation (2.4) we need only consider values of m' that are at most the number of vertices in $A(t)$ since all other values of $C_{A(t)}(\cdot)$ are infinite. This means that we can amortise the runtime needed to compute all values of $C_{\mathcal{P}(S)}(\cdot)$ for a particular IFS S to $\mathcal{O}(\sum_{P \in \mathcal{P}(S)} n|P|) = \mathcal{O}(n^2)$. This is true because we need to consider at most all the $n+1$ possible values of m for each $P \in \mathcal{P}(S)$ while the parts in $\mathcal{P}(S)$ are disjoint. To compute all values of $C_P(\cdot)$ for all segments in $\mathcal{S}$ we need to compute all values of $C_{\mathcal{P}(S^*)}(\cdot)$ for all IFSs in all IFS covering sets for all segments in $\mathcal{S}$. Hence computing the whole table for all values of $C_{\mathcal{P}(S^*)}(\cdot)$ takes $\mathcal{O}(n^2 \cdot \sum_{s \in \mathcal{S}} |\mathcal{K}_{A(s)} \cup \mathcal{K}_{B(s)}|)$ time. Therefore the runtime of the algorithm (Figure 2.5) is dominated by the time needed to compute the table containing the values of the functions $C_{\mathcal{P}(S^*)}(\cdot)$.

2.4 Counting Segments and IFS Covering Sets

In Section 2.3 we have seen that we can efficiently compute optimal m-cuts for solid grids if the number of considered segments and IFS covering sets is small. Hence we need to identify

a small IFS covering set $\mathcal{K}_P$ for each considered cut out part $P \in \{A(s), B(s)\}$ and $s \in \mathcal{S}$. In this section we will prove that the runtime of the given algorithm is $\mathcal{O}(n^4)$ as claimed, by counting the number of segments and the sizes of the IFS covering sets. In order for the involved sets not to be too large, the set $\mathcal{S}$ includes only straight, and corner segments, together with the stair, clamp, and square segments without breaks. It also contains all stair segments that consist of only a broken bar. Additionally $\mathcal{S}$ contains all stair, clamp, and square segments with breaks that have at most $c\sqrt{n}$ edges, for some constant c according to Theorem 2.7. The latter theorem together with Lemma 2.5 guarantees that these sets $\mathcal{C}$ and $\mathcal{S}$ suffice in order to compute an optimal m-cut using the algorithm in Section 2.3.

According to the results in Section 2.3 we need to show that $\sum_{s\in\mathcal{S}} |\mathcal{K}_{A(s)} \cup \mathcal{K}_{B(s)}| \in \mathcal{O}(n^2)$ and that all required segments and IFS covering sets can be found efficiently, in order to guarantee a total runtime of $\mathcal{O}(n^4)$. We start by counting the number of segments for each segment type.

Lemma 2.10. *There are $\mathcal{O}(n)$ straight and corner segments, and $\mathcal{O}(n)$ stair segments consisting of only a broken bar. Also there are $\mathcal{O}(n^2)$ stair, clamp, and square segments without breaks, and $\mathcal{O}(n^2)$ of these segments with breaks having a length of at most $c\sqrt{n}$. Furthermore all of these segments can be enumerated in time $\mathcal{O}(n^2)$.*

Proof. According to Definition 2.4 a straight segment does not have any bends. We can identify such a segment with one of its edges that touches the exterior face of the grid graph. Since the grid has $\mathcal{O}(n)$ edges there are also at most $\mathcal{O}(n)$ straight segments. A corner segment on the other hand can be identified by its bend and the direction into which it points. Since there are four possible directions and $\mathcal{O}(n)$ faces in the grid graph that can be bends, there are $\mathcal{O}(n)$ corner segments.

There are $\mathcal{O}(n^2)$ stair, clamp, and square segments without breaks since according to Definition 2.4 each such segment can be identified with the respective bend f_2, the directions in which f_2 points (the directions in which the other bends point is determined by this), and a distance to the consecutive bends of

f_2 (which, in the case of a square segment, amounts to choosing the length of the bar b and one of two possible lengths for b'). Since there are $\mathcal{O}(n)$ faces in the grid graph that can be used for f_2, four directions in which to point, and the distance can be at most n, the result follows.

For segments with breaks each choice of the above three parameters also leaves the choice of a position of the break along the length of the respective broken bar. If the segment is a stair segment consisting of only a broken bar then the only choice is its break and the direction in which the bends point. Thus there are only $\mathcal{O}(n)$ many such segments. For all other segments with breaks the length of the segment is assumed to be at most $c\sqrt{n}$. Therefore there are only $\mathcal{O}(\sqrt{n})$ choices for the distance between the bends. This also means that there are $\mathcal{O}(\sqrt{n})$ possible positions for the break, and hence the total number of segments with breaks is again $\mathcal{O}(n^2)$.

The above counting arguments clearly give a straightforward way of enumerating all these segments too. Hence they can be found in time $\mathcal{O}(n^2)$. □

Next we identify the IFS covering sets used for each of the segments $s \in \mathcal{S}$ and their cut out parts. We will prove that there is an IFS covering set of constant size for each segment in $\mathcal{S} \setminus \mathcal{C}$, and a linear sized set for each in $\mathcal{C}$. The given construction of the sets can be used to compute them in a preprocessing step of the algorithm in $\mathcal{O}(n^2)$ total time.

In the following we will count the number of IFSs in the IFS covering sets for every type of segment separately. Given a segment s of a certain type and $P \in \{A(s), B(s)\}$ we will construct a covering set $\mathcal{K}_P$. According to Definition 2.9, for each case we need to show that for any IFS $S \subseteq \mathcal{C}_P$ there exists a set $S^* \in \mathcal{K}_P$ for which $S \subseteq \mathcal{C}_{\mathcal{P}(S^*)}$. In all cases, if $\mathcal{C}_P = \emptyset$ then $\mathcal{K}_P = \emptyset$ is an IFS covering set. Hence in the following we assume that $\mathcal{C}_P \neq \emptyset$. To prove our claims we need to give the respective segments an orientation in the given grid graph G. Therefore we define *horizontal* respectively *vertical* edges of G to be those for which the incident vertices have the same

y- respectively x-coordinates. A *horizontal* bar of a segment contains only vertical edges, and a *vertical* bar only horizontal edges. A *horizontal* respectively *vertical* broken bar of a segment contains two horizontal respectively vertical bars.

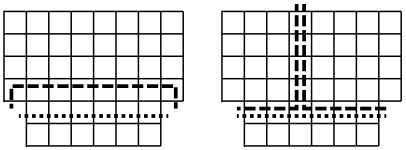
In all but the case of square segments with breaks we will ignore the fact that some of the segments in the claimed IFS covering set may partially lie outside of the grid graph in the following proofs. The reason why we can ignore these special cases is that if any segment, except for a clamp or square segment with a break, is split at any point (by the border of the grid graph) then the resulting segments that lie inside the grid graph are all segments of types again included in $\mathcal{S}$. Hence these can be used for the IFS. On the other hand, splitting a clamp or square segment s with a break may not result in a set of segments from $\mathcal{S}$. For clamp segments our choices of the IFSs will however never result in segment types not in $\mathcal{S}$. We will therefore only need to handle the case of square segments separately.

We begin with the segments in $\mathcal{C}$ and show that their IFS covering sets are linear sized. As the next lemma shows the case of straight segments and the convex part cut out by a corner segment are very similar.

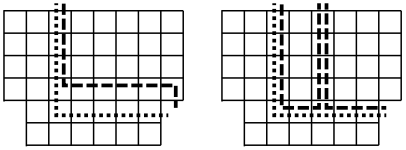
Lemma 2.11. *Consider a straight or corner segment $s \in \mathcal{C}$ in a solid grid graph G. Let in the latter case $P \in \{A(s), B(s)\}$ such that P is the convex part cut out by the corner segment, and $P \in \{A(s), B(s)\}$ arbitrary otherwise. There is an IFS covering set $\mathcal{K}_P$ containing $\mathcal{O}(n)$ IFSs.*

Proof. If s is a straight segment (Figure 2.6(a)) assume w.lo.g. that s is a horizontal bar and that the upper vertices incident to the edges of s are those belonging to P. If s is a corner segment (Figure 2.6(b)) we assume w.l.o.g. that its bend points up and right. Let in both cases b be the horizontal bar of s, i.e. if s is a straight segment then $b = s$.

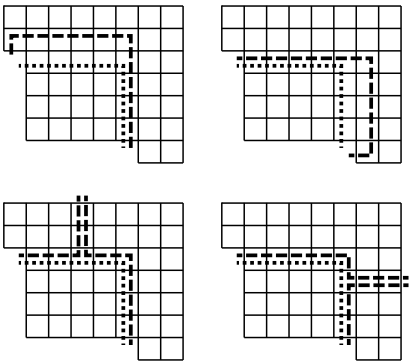
Let $T \subseteq \mathcal{C}_P \setminus \{s\}$ contain all segments from $\mathcal{C}_P$ different from s that include edges from b. Since any straight segment including edges from b is either s itself or cuts out vertices not in P, the segments in T are all corner segments with bends that are



(a) straight segment



(b) corner segment: convex part



(c) corner segment: concave part

Figure 2.6: *The IFS covering sets for straight and corner segments s. The segment s is indicated by the dotted line and the segments in an IFS by the dashed lines.*

interior faces touching two edges of b. Let $t \in \mathcal{S}$ be the segment cutting out all vertices from P except those incident to b. If s is a straight segment t is a clamp segment without break, a corner, or straight segment. If s is a corner segment, depending on the length of b, t is either a corner or a stair segment without break. For the IFS $S_1^* = \{t\}$ in P it holds that $\mathcal{C}_{\mathcal{P}(S_1^*)} = \mathcal{C}_P \setminus (T \cup \{s\})$ since the only edges that can not be used to form a segment from $\mathcal{C}_{\mathcal{P}(S_1^*)}$ are those in b. Hence for any IFS $S \subseteq \mathcal{C}_P$ which does not contain a segment from T it holds that $S \subseteq \mathcal{C}_{\mathcal{P}(S_1^*)}$. Let $u \in T$ be a corner segment cutting out $A(u)$ from P. All vertices in $P \setminus A(u)$ are cut out by a segment u' which is a corner segment if s is a straight segment. If s is a corner segment then u' is a clamp segment without break. Consider the case when an IFS $S \subseteq \mathcal{C}_P$ contains u. Since the vertices not cut out by u from P are cut out by u' we know that $S \subseteq \mathcal{C}_{\mathcal{P}(S_u^*)}$, where $S_u^* = \{u, u'\}$. Hence the set $\mathcal{K}_P$ containing S_1^* and the S_u^* for all segments $u \in T$ is an IFS covering set. Since by Lemma 2.10 there are $\mathcal{O}(n)$ segments in T the size of $\mathcal{K}_P$ is also $\mathcal{O}(n)$. □

Concerning segments in $\mathcal{C}$ we are left with the case when the cut out part of a corner segment is concave. The following lemma handles this case.

Lemma 2.12. *For any corner segment $s \in \mathcal{C}$ in a solid grid graph G and $P \in \{A(s), B(s)\}$ such that P is the concave part cut out by s, there is an IFS covering set $\mathcal{K}_P$ containing $\mathcal{O}(n)$ IFSs.*

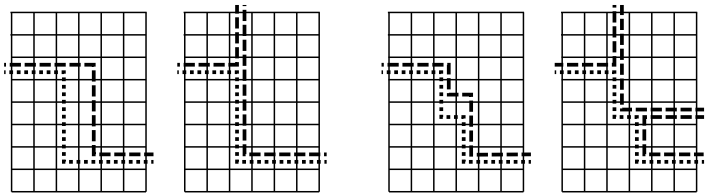
Proof. Assume w.l.o.g. that the bend of s points left and down (Figure 2.6(c)). We denote the horizontal bar of s by b_1 and the vertical bar by b_2. Let for $j \in \{1, 2\}$ the set $T_j \subseteq \mathcal{C}_P \setminus \{s\}$ contain all segments from $\mathcal{C}_P$ different from s that include edges from b_j. There is a corner or clamp segment without break $t_j \in \mathcal{S}$ that cuts out all vertices from P except those that are incident to edges in b_j. For the IFS $S_j^* = \{t_j\}$ in P it holds that $\mathcal{C}_{\mathcal{P}(S_j^*)} = \mathcal{C}_P \setminus (T_j \cup \{s\})$ since the only edges that can not be used to form a segment from $\mathcal{C}_{\mathcal{P}(S_j^*)}$ are those in b_j. Hence for any IFS $S \subseteq \mathcal{C}_P$ which does not contain a segment from T_j it holds that $S \subseteq \mathcal{C}_{\mathcal{P}(S_j^*)}$.

We are left with the case when an IFS $S \subseteq \mathcal{C}_P$ in P both contains a segment u_1 from T_1 and u_2 from T_2 (note that $T_1 \cap T_2 = \emptyset$ since these sets only contain straight and corner segments different from s). One of these two segments, say u_1, must have a bend f that is a face touched by two edges of s. Otherwise u_1 and u_2 would cross at the bend of s which is a contradiction to the fact that S is non-crossing. There is a corner segment $r \in T_1$ that has f as its bend which for r points left and up. Note that r may or may not be equal to u_1 since u_1 may point left or right. In any case however, r and u_1 share the same vertical bar, while the horizontal bar of r is part of b_1. This means that no segment in S crosses r. We can thus conclude that any segment in S either cuts out vertices from $A(r)$ or from $P \setminus A(r)$, where $A(r)$ is the part cut out by r from P. The vertices in $P \setminus A(r)$ are cut out by either a straight or a stair segment without break $r' \in \mathcal{S}$, depending on whether the horizontal bar of r is equal to b_1 or not. Hence for the IFS S containing u_1 it holds that $S \subseteq \mathcal{C}_{\mathcal{P}(S_r^*)}$ where $S_r^* = \{r, r'\}$. In case the bend f belongs to u_2 of S we can analogously construct an IFS for a corner segment $r \in T_2$ which also has f as its bend and for which the vertical bar is part of b_2. Accordingly there is a straight or stair segment without break $r' \in \mathcal{S}$ such that $S \subseteq \mathcal{C}_{\mathcal{P}(S_r^*)}$ where $S_r^* = \{r, r'\}$. Hence we can conclude that the set $\mathcal{K}_P$ containing S_1^*, S_2^*, and S_r^* for the respective corner segments $r \in T_1 \cup T_2$ is an IFS covering set. Since by Lemma 2.10 there are $\mathcal{O}(n)$ corner segments, the size of $\mathcal{K}_P$ is also $\mathcal{O}(n)$. □

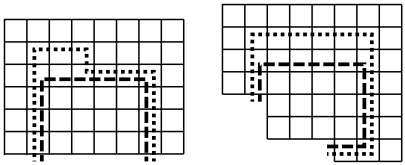
We now turn to segments in $\mathcal{S} \setminus \mathcal{C}$ for which we need to show constant sized IFS covering sets. We begin with the stair segments.

Lemma 2.13. *For any stair segment $s \in \mathcal{S}$ in a solid grid graph G and any $P \in \{A(s), B(s)\}$ there is an IFS covering set $\mathcal{K}_P$ containing two IFSs.*

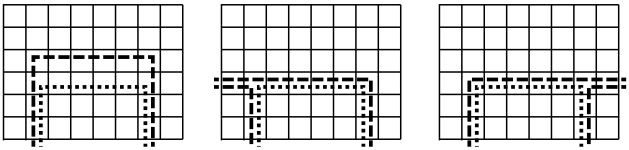
Proof. If s has no break let b be the bar of s ending at its two bends. Otherwise b is the broken bar of s. We assume w.l.o.g. that any bend of s points either down and left, or up and right. Furthermore b is assumed to be vertical, and the vertices that



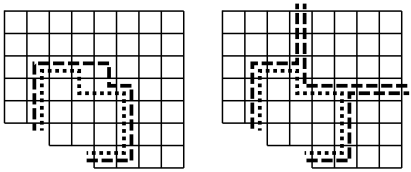
(a) stair without break (b) stair with break



(c) clamp, square: convex part



(d) clamp without break: concave part



(e) square with break: concave part

Figure 2.7: *The IFS covering sets for stair, clamp, and square segments s. The segment s is indicated by the dotted line and the segments in an IFS by the dashed lines.*

are the upper and right incident vertices of the vertical and horizontal edges of s, respectively, are those belonging to P.

If s has no break (Figure 2.7(a)) it has two bends and hence any straight segment including edges from b would cut out vertices not contained in P. Thus any segment from $\mathcal{C}_P$ that includes edges from b must be a corner segment. Let $T \subseteq \mathcal{C}_P$ contain all these corner segments. Depending on the lengths of the horizontal bars of s, there is either a corner or a stair segment without break $t \in \mathcal{S}$ that cuts out all vertices from P except those that are incident to the edges in b. For the IFS $S_1^* = \{t\}$ in P it holds that $\mathcal{C}_{\mathcal{P}(S_1^*)} = \mathcal{C}_P \setminus T$ since the only edges that can not be used to form a segment from $\mathcal{C}_{\mathcal{P}(S_1^*)}$ are those in b. Hence for any IFS $S \subseteq \mathcal{C}_P$ which does not contain a segment from T it holds that $S \subseteq \mathcal{C}_{\mathcal{P}(S_1^*)}$. Let $u \in \mathcal{S}$ be the corner segment that cuts out a part $A(u)$ from P and has the same bend as s, say f_1, pointing up and right. The vertices in $P \setminus A(u)$ are cut out by the corner segment $u' \in \mathcal{S}$ having the other bend f_2 of s as its bend which for u' however points up and left. The only segments from $\mathcal{C}_P$ that are not included in $\mathcal{C}_{\mathcal{P}(S_2^*)}$, where $S_2^* = \{u, u'\}$, are those that have a horizontal bar crossing the vertical bars of u and u' above the bend f_2 of s. The vertical bar of any $r \in T$ is included in the vertical bar of u. Hence if an IFS $S \subseteq \mathcal{C}_P$ contains a corner segment $r \in T$ then $S \subseteq \mathcal{C}_{\mathcal{P}(S_2^*)}$ since S is non-crossing. In conclusion the set $\mathcal{K}_P$, for any $P \in \{A(s), B(s)\}$, is an IFS covering set if it contains the IFSs S_1^* and S_2^* in case s is a stair segment without break.

If s is a stair segment with break (Figure 2.7(b)) let v be the vertex in the cut out part P incident to the break of s. Two of the four incident edges to v, say e_1 and e_2, belong to s, while the other two, say e_1' and e_2', are not included in s. Depending on the lengths of the bars of s, the segment t containing the edges of s but where e_1 and e_2 are exchanged with e_1' and e_2' is either a corner, or a stair segment. Note that t may or may not have a break if it is a stair segment, but t is always included in $\mathcal{S}$ since it contains the same number of edges as s. The only segment in $\mathcal{C}_P$ that cuts out the vertex v from P is the corner segment r having the face touching e_1 and e_2 as its bend and pointing up and right. Hence if an IFS $S \subseteq \mathcal{C}_P$ does not contain r then

$S \subseteq \mathcal{C}_{\mathcal{P}(S_1^*)}$, where $S_1^* = \{t\}$. The vertices from P that are not contained in the cut out part $A(r) \subset P$ by r are cut out by the following segments. Consider those belonging to the connected component induced by $P \setminus A(r)$ and containing vertices incident to the edges of the horizontal bar of r. Depending on the lengths of the horizontal bars of s, these vertices are cut out by a corner or clamp segment without break $u \in \mathcal{S}$. All remaining vertices in P, if any, are cut out by a corner segment $u' \in \mathcal{S}$. Let the IFS S_2^* contain r and u, and also u' if it exists. Obviously any IFS S containing r must be contained in the set $\mathcal{C}_{\mathcal{P}(S_2^*)}$. Therefore in case s is a stair segment with break the set $\mathcal{K}_P$, for any $P \in \{A(s), B(s)\}$, containing the IFSs S_1^* and S_2^* is an IFS covering set. □

As the next lemma shows, the clamp and square segments are similar. We begin with the convex cut out part.

Lemma 2.14. *For any clamp or square segment $s \in \mathcal{S}$ in a solid grid graph G and $P \in \{A(s), B(s)\}$ such that P is the convex part cut out by s, there is an IFS covering set $\mathcal{K}_P$ containing one IFS.*

Proof. We use the same names for the bends (f_1, f_2, f_3) and (broken) bars (b, b') of s as in Definition 2.4. Assume w.l.o.g. that the bend f_1 points down and right (Figure 2.7(c)). Let $b'' \subseteq b$ be the horizontal bar of s ending at f_1. Any straight or corner segment including edges from b'' cuts out vertices that are not included in P since such a segment has at most one bend. Hence for the segment t cutting out all vertices from P except those that are incident to the edges in b'' it holds that $\mathcal{C}_{\mathcal{P}(S^*)} = \mathcal{C}_P$, where $S^* = \{t\}$. Depending on the lengths of the vertical bars of s, note that t is either a straight, corner, or clamp segment if s is a clamp segment, and that t is a clamp or square segment if s is a square segment. Furthermore if t is a clamp or square segment it does not have a break, regardless of whether s has one. If t is a square segment it is also important to check that the horizontal and vertical bars of t not ending at the exterior face of G have the correct lengths. This is true due to the assumptions on the sizes of s's bars b and b' according to Definition 2.4, except in the case when s is a square segment

without break and b' is smaller than b. However in the latter case we may consider the analogous case where the identities of b and b' are swapped. Hence $t \in \mathcal{S}$ which means that in this case $\mathcal{K}_P = \{S^*\}$ is an IFS covering set. □

Next we turn to the concave cut out part of a clamp or square segment. The following lemma handles the case when the segment has no break.

Lemma 2.15. *For any clamp or square segment $s \in \mathcal{S}$ without break in a solid grid graph G and $P \in \{A(s), B(s)\}$ such that P is the concave part cut out by s, there is an IFS covering set $\mathcal{K}_P$ containing three IFSs.*

Proof. As in the proof of Lemma 2.14 we use the same names for bends and bars as in Definition 2.4 and assume that f_1 points down and right (Figure 2.7(d)). Let $T \subseteq \mathcal{C}_P$ contain all straight and corner segments that include some edge from the bar b. There is a clamp or square segment t without break (t is of the same type as s) that cuts out all vertices in P except those that are incident to the edges in b. For the IFS $S_1^* = \{t\}$ in P it holds that $\mathcal{C}_{\mathcal{P}(S_1^*)} = \mathcal{C}_P \setminus T$ since the only edges that can not be used to form a segment from $\mathcal{C}_{\mathcal{P}(S_1^*)}$ are those in b. Hence for any IFS $S \subseteq \mathcal{C}_P$ which does not contain a segment from T it holds that $S \subseteq \mathcal{C}_{\mathcal{P}(S_1^*)}$. Let t_1 be the corner segment that has the bend f_1 of s as its bend which for t_1 points left and down. Also let t_1' be the corner segment with a bend at f_1 that for t_1' however points right and down. In case s is a clamp segment let t_2 be the corner segment that has the bend f_2 of s as its bend which for t_2 points left and down. Also let t_2' be the corner segment with a bend at f_2 which for t_2' points right and down. If s is a square segment we define t_2 to be the clamp segment without break that has the bends f_2 and f_3 of s as bends such that f_2 points left and down for t_2. In addition we define t_2' to be a stair segment that has f_2 and f_3 as its only bends such that f_2 points right and down for t_2'. Depending on the length of the bar b' of s, the stair segment t_2' either has no break or consists of only a broken bar. Thus all of the defined segments above are contained in $\mathcal{S}$. For any of the cases the only segments from $\mathcal{C}_P$ that are not included in $\mathcal{C}_{\mathcal{P}(S_2^*)}$, where $S_2^* = \{t_1, t_2\}$, are those that have a vertical bar

crossing the horizontal bars of t_1 and t_2 to the left of the bend f_1 of s. Analogously the only segments from $\mathcal{C}_P$ that are not included in $\mathcal{C}_{\mathcal{P}(S_3^*)}$, where $S_3^* = \{t_1', t_2'\}$, are those that have a vertical bar crossing the horizontal bars of t_1 and t_2 to the right of the bend f_2 of s. Any straight or corner segment $r \in T$ that contains an edge from b has a horizontal bar that extends to the left from f_1 or to the right from f_2, since r has at most one bend. Hence if an IFS $S \subseteq \mathcal{C}_P$ contains a segment $r \in T$ for which the horizontal bar is included in the horizontal bar of t_2, since S is non-crossing it must be that $S \subseteq \mathcal{C}_{\mathcal{P}(S_2^*)}$. Analogously, if the horizontal bar of r is included in the horizontal bar of t_1' then $S \subseteq \mathcal{C}_{\mathcal{P}(S_3^*)}$. Therefore in this case the set $\mathcal{K}_P = \{S_1^*, S_2^*, S_3^*\}$ is an IFS covering set. □

The case when a clamp or square segment has a break and the concave cut out part is considered is the most complicated one. This is because the claimed segments in the IFS covering set may be split by the border of the grid graph and the resulting segments might not be segments from the set $\mathcal{S}$. Hence we need to handle this case separately in the proof of the following lemma.

Lemma 2.16. *For any clamp or square segment $s \in \mathcal{S}$ with break in a solid grid graph G and $P \in \{A(s), B(s)\}$ such that P is the concave part cut out by s, there is an IFS covering set $\mathcal{K}_P$ containing at most three IFSs.*

Proof. As in the proof of Lemmas 2.14 and 2.15 we use the same names for bends and bars as in Definition 2.4 and assume that f_1 points down and right (Figure 2.7(e)). Recall that if s is a square segment having a break then f_1 and its consecutive bend touching the break point in a common direction. Let v be the vertex in the concave cut out part P incident to the break of s. Two of the four incident edges to v, say e_1 and e_2, belong to s, while the other two, say e_1' and e_2', are not included in s. Let t be the clamp or square segment (t is of the same type as s) that contains the edges of s but where e_1 and e_2 are exchanged with e_1' and e_2'. Note that t may or may not have a break, but t is always included in $\mathcal{S}$ since it contains the same number of edges as s. The only segment in $\mathcal{C}_P$ that cuts out the vertex v from P

is the corner segment r having the face touching e_1 and e_2 as its bend which points up and right. Hence if an IFS $S \subseteq \mathcal{C}_P$ does not contain r then $S \subseteq \mathcal{C}_{\mathcal{P}(S_1^*)}$, where $S_1^* = \{t\}$. The vertices from P that are not contained in the cut out part $A(r) \subset P$ by r are cut out by the following segments. Those belonging to the connected component induced by $P \setminus A(r)$ and containing vertices incident to the edges of the horizontal bar of r are cut out by a corner or stair segment $u \in \mathcal{S}$, if s is a clamp or square segment respectively. If u is a stair segment it either has no break or it consists of only a broken bar, depending on the size of b'. All other vertices in P are cut out by a stair segment $u' \in \mathcal{S}$. Also u' either has no break or consists of only a broken bar, depending on the size of b. Obviously any IFS S containing r must be contained in the set $\mathcal{C}_{\mathcal{P}(S_2^*)}$, where $S_2^* = \{r, u, u'\}$. Therefore in this case the set $\mathcal{K}_P = \{S_1^*, S_2^*\}$ is an IFS covering set.

Notice that in the last case, if s is a square segment with break the edges e_1' and e_2' may touch the exterior face. In this case the segment t is split at the corresponding face (Figure 2.8). If t is a square segment with break, one of the resulting segments however is of none of the types that are contained in $\mathcal{S}$. This means that for this special case we need to find a different IFS covering set than proposed above. The set S_2^* can still be used and thus for any IFS $S \subseteq \mathcal{C}_P$ containing r it holds that $S \subseteq \mathcal{C}_{\mathcal{P}(S_2^*)}$. However for the case when $r \notin S$, instead of S_1^* as above we construct the following two IFSs for $\mathcal{K}_P$. Let e_1' be horizontal and e_2' vertical. Let $v' \in P$ be the vertex not equal to v incident to e_1'. Apart from r there is only one other segment $r' \in \mathcal{C}_P$ that cuts out v' from P, which as r is also a corner segment. Except for v, the vertices from P that are not contained in the cut out part $A(r') \subset P$ by r' are cut out by the following segments. Those belonging to the connected component induced by $P \setminus (A(r') \cup \{v\})$ and containing vertices incident to the edges of the horizontal bar of r' are cut out by a stair segment $u \in \mathcal{S}$. This stair segment either has no break or consists of only a broken bar, depending on the size of b'. All remaining vertices in $P \setminus \{v\}$ are cut out by a corner segment u'. Obviously any IFS $S \subseteq \mathcal{C}_P$ not containing r but r' must be contained in the set $\mathcal{C}_{\mathcal{P}(S_3^*)}$, where $S_3^* = \{r', u, u'\}$. All vertices

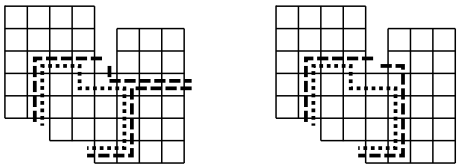


Figure 2.8: *The additional IFSs in $\mathcal{K}_P$ if s is a square segment with break and the border of the grid would split one of the segments in an IFS.*

in P except v and v' that are not contained in the cut out part $A(u') \subset P$ by u' are cut out by a clamp segment u'' without break. Hence for an IFS $S \subseteq \mathcal{C}_P$ neither including r nor r' it must be that $S \subseteq \mathcal{C}_{\mathcal{P}(S_4^*)}$, where $S_4^* = \{u', u''\}$. Therefore in this special case the set $\mathcal{K}_P = \{S_2^*, S_3^*, S_4^*\}$ is an IFS covering set. □

All constructions in the above lemmas give a straightforward way to enumerate the IFS covering sets. Since by Lemma 2.10 there are $\mathcal{O}(n^2)$ many segments in $\mathcal{S} \setminus \mathcal{C}$ and each such segment needs at most three IFS covering sets, all these sets can be found in time $\mathcal{O}(n^2)$. For segments in $\mathcal{C}$ Lemma 2.10 asserts that there are $\mathcal{O}(n)$ many while the IFS covering sets also have $\mathcal{O}(n)$ size. Hence also these sets can be found in time $\mathcal{O}(n^2)$. By the same counting arguments it also follows that the total number $\sum_{s \in \mathcal{S}} |\mathcal{K}_{A(s)} \cup \mathcal{K}_{B(s)}|$ of IFS covering sets that need to be considered by the algorithm in Section 2.3 is $\mathcal{O}(n^2)$. Hence the theorem stated below follows from these observations.

Theorem 2.17. *For any m the cut size of an optimal m-cut in a solid grid graph can be computed in time $\mathcal{O}(n^4)$.*

2.5 Generalisations and Faster Algorithms

We have seen that for solid grid graphs the cut size of an optimal m-cut can be computed in $\mathcal{O}(n^4)$ time. Clearly the presented

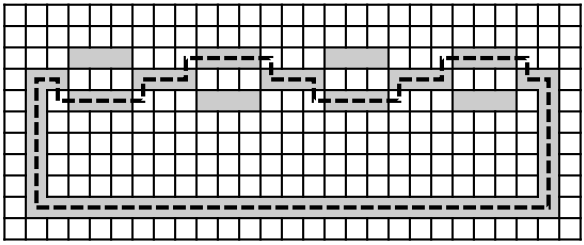


Figure 2.9: *A grid graph with holes (highlighted in grey). The optimum solution requires a segment (dashed lines) that alternates between the lower and upper paths through the holes. This pattern indicates that there can be an exponential number of segments that have to be considered.*

dynamic programming scheme can also be used to compute an optimal m-cut by storing the segments used in each recursive step. From this set of segments the corresponding vertex partition can also easily be computed. Hence the BISECTION problem can be solved in $\mathcal{O}(n^4)$ time on solid grid graphs. This improves on the formerly fastest known algorithm by Papadimitriou and Sideri [54] which has a runtime of $\mathcal{O}(n^5)$.

The main reason why the BISECTION problem can be solved in polynomial time is that all segments that come into question for an optimal solution can be exhaustively searched. Unfortunately this also means that generalising the used techniques to grid graphs with holes does not immediately yield an efficient algorithm. This is because for this graph class one can construct examples for which an exponential number of segments seem to come into question for an optimal solution (Figure 2.9). This also means that the presented method does not immediately give a key to resolving the long standing open problem of determining the complexity of BISECTION on planar graphs. Another case where too many segments exist for an efficient algorithm of the sort presented in this chapter, is when the edges in a solid grid can have weights. This is because holes can be simulated by small edge weights (cf. Figure 2.9). Vertex weights on the other hand may render the problem weakly NP-hard.

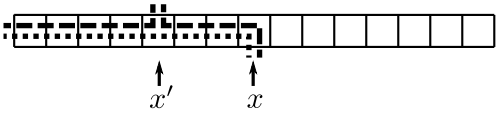


Figure 2.10: *This solid grid graph has constant height and linear width. For each position x along its width there is a corner segment with an IFS for each position x' left of x in its IFS covering set (Lemma 2.12). For each value of m the algorithm combines the solutions computed for the two segments in the IFS. To do this it needs to consider every suitable value m' (Equation (2.4)). This gives a total asymptotic runtime of $\Omega(n^4)$.*

However for other graph classes similar observations on the number of segments and IFS covering sets as the ones presented in this chapter should be achievable. These include graphs that correspond to regular tessellations of the plane. For instance in 2D finite element models, apart from the tessellations into quadrilaterals that we chose as a model, triangulations are also used [19, 62]. For these graphs polynomial time algorithms that optimally solve the `BISECTION` problem should exist. However for practical purposes the runtime of $\mathcal{O}(n^4)$ is too slow. Can faster algorithms be found to solve the problem?

One major inspiration to the algorithm presented in this chapter comes from the corresponding one by MacGregor for trees [47]. In particular, combining precomputed solutions using the min-convolution for an IFS (as in Equation (2.4) page 28) is the same in both algorithms and needs $\mathcal{O}(n^2)$ time. Since a tree has a linear number of segments the runtime of $\mathcal{O}(n^3)$ follows for the corresponding algorithm. However the runtime of this algorithm was later found [32] to actually be $\mathcal{O}(n^2)$ using a more sophisticated amortised runtime analysis. Hence it suggests itself to also try similar techniques on our algorithm for grids. The only additional layer that our algorithm adds to the recursive scheme of the algorithm for trees, is going through the IFSs in the IFS covering sets. In a tree each of the $n-1$ segments (i.e. edges) can be seen as having a trivial IFS covering set of size one. The only IFS in such a set contains those edges as segments that

cut off the subtrees below the currently considered one. Since we show in Section 2.4 that for solid grid graphs the total number of IFS covering sets is $\mathcal{O}(n^2)$, it seems as if the runtime of the grid algorithm should only be a linear factor worse than that of the tree algorithm. That is, we hope for an actual runtime of $\mathcal{O}(n^3)$ for our algorithm. Unfortunately one can find examples (Figure 2.10) of grid graphs in which the algorithm needs $\mathcal{O}(n^4)$ time, which settles this matter.

Hence the question remains whether faster algorithms to solve the `BISECTION` problem on solid grid graphs can be found. The used techniques for the algorithm in this chapter do not seem to yield any faster methods. However if one is willing to settle with approximate solutions the next chapter shows how to compute solutions faster.

Chapter 3

Corner Cuts and their Applications

This chapter is concerned with approximately solving the BISECTION[1] problem on solid grid graphs[2]. In the first part of this chapter we will show that a cut containing only simple shaped segments[3] is a good approximation to the optimum. More precisely, restricting the cuts to contain only straight and corner segments[4] we can cut out a number of vertices arbitrarily close to a given number m, while the cut size is still close to the optimum of an m-cut[5]. In the second part of this chapter we will show how this fact can be used algorithmically. We use known techniques to yield a fast bicriteria approximation algorithm for the BISECTION problem. In particular we show that we can compute a solution for which the cut out parts are arbitrarily close to $\lceil n/2 \rceil$, while the cut size deviates only by a constant factor from the optimum. This can be done in $\mathcal{O}(n^{1.5})$ time on solid grid graphs. Additionally we are able to harness the used techniques to yield a fast bicriteria approximation algorithm for

[1] Definition 2.1 page 12
[2] Definition 1.2 page 4
[3] Definition 2.2 page 12
[4] Definition 2.4 page 17
[5] Definition 2.3 page 16

the k-BALANCED PARTITIONING[6] problem. Here the sizes of the parts are at most twice $\lceil n/k \rceil$, while the cut size is within a logarithmic factor from optimum. The runtime is $\mathcal{O}(n^{1.5} \log k)$ for solid grid graphs.

The results in the first part of this chapter (Section 3.2) were obtained in collaboration with Shantanu Das and Peter Widmayer and were published as an extended abstract [23]. The second part (Section 3.3) is the sole work of the author of this dissertation and was published in [21].

3.1 A Failed Attempt and a Solution by Detours

The first part of this chapter aims at understanding the intricacies of optimally cutting out a fixed number m of vertices in a graph from a novel point of view: we study simple cut shapes, for which it can be shown that they compare well to optimal unrestricted ones. In related problems on polygons similar ideas have led to interesting insights in the past. For instance *guillotine cuts* have been considered, which are orthogonal straight-lined cuts. When an orthogonal polygon is to be partitioned into rectangles these lead to good approximations [10]. Also if a rectangular polygon is to be partitioned into rectangles fulfilling given size constraints, good solutions can be achieved by using guillotine cuts [36]. In our setting a guillotine cut in a solid grid graph corresponds to a set of straight segments. However these kinds of cuts do not yield satisfactory solutions since they can be far away from optimum (Figure 3.1). On the other hand, as observed in the previous chapter (Theorem 2.6 page 19) there always exists an optimum solution to cut out m vertices in which almost all segments have a simple shape. More precisely, at most one segment in an optimal solution is not a straight or corner segment.

The above two observations on the shapes of segments and their relation to optimal solutions naturally lead to the question

[6] Definition 1.1 page 3

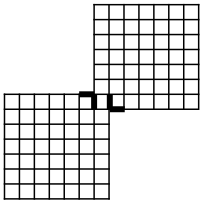


Figure 3.1: *A grid in which the optimum cut size for $m = n/2$ is constant (bold edges) but any cut containing only straight segments has cut size $\Omega(\sqrt{n})$.*

of how well *corner cuts* perform. These contain only straight and corner segments (Figure 3.2). In the first part of this chapter (Section 3.2) we prove that optimum corner cuts get us arbitrarily close to a cut out part of size m, and that this limitation makes us lose only a small factor in the quality of the solution. More precisely, we show that for an optimal m-cut with cut size C^* in a solid grid graph and any $\varepsilon \in]0, 1]$, there exists a corner m'-cut for some $m' \in [(1-\varepsilon)m, (1+\varepsilon)m]$. Furthermore its cut size deviates by at most a factor of $\mathcal{O}(1/\sqrt{\varepsilon})$ from C^*. We achieve our result by proving a number of theorems for polygons that we relate to the case of grid graphs. The reason for choosing this approach is that polygons are continuous objects which is in contrast to the discrete nature of graphs. This fact makes certain tools available for our proofs that otherwise would not be applicable. The first part of this chapter will therefore be concerned with thoroughly analysing corner cuts in polygons.

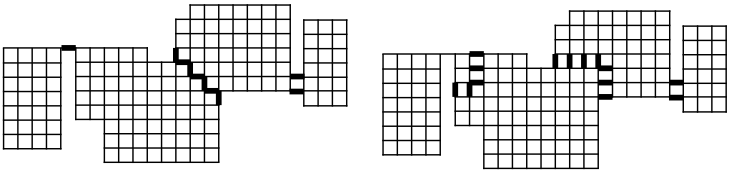


Figure 3.2: *An optimal (left) and a corner cut (right) in a solid grid graph, each cutting out $m = 110$ vertices. The bold edges indicate the edges of the segments.*

A first attempt at finding approximate solutions to the BISECTION problem using the above result would be to find an algorithm that will compute optimal corner cuts. We showed [22] that such an algorithm exists. More precisely, this algorithm is the same as the one presented in Section 2.3 but considers only straight and corner segments. In particular the second phase (cf. Figure 2.5 page 29) is omitted. Since the algorithm will compute the optimum corner m-cut for any $m \in \{0, \ldots n\}$, the best solution in the interval $[(1-\varepsilon)m, (1+\varepsilon)m]$ can be computed for any $\varepsilon \in]0, 1]$. Hence the output will be a good approximation to the optimal m-cut. The paper presented in [22] goes on to show that IFS covering sets for straight and corner segments can be constructed using only these types of segments. However the sizes of the resulting IFS covering sets are linear and thus the runtime of the algorithm from [22] is $\mathcal{O}(n^4)$ (cf. Section 2.4). Note that the latter algorithm is faster than the one presented in Section 2.3 since it omits the second phase. However the asymptotic runtime is the same and is thus too slow for practical purposes. Hence no useful approximation algorithm is obtained.

The question of how corner cuts can be used algorithmically thus remains. To answer this question we take a detour and consider other problems in which a graph is to be cut into two parts with an additional constraint on the sizes of the resulting parts. For such problems the size constraint is realised by demanding the fulfilment of a bound on the size m of one of the parts, or the optimisation function may also depend on m. Apart from the BISECTION problem, two examples of these types of problems include the EDGE SEPARATOR problem in which $bn \leq m \leq (1-b)n$ for a given value $b \leq 1/3$, and the SPARSEST CUT problem in which the function $\frac{C}{m(n-m)}$ is to be optimised. In the latter, C denotes the cut size of the solution. Note that in any of these problems the respective optimal solution cuts out m vertices using a minimum number of edges for this particular value of m. Hence in a planar graph it constitutes an optimal m-cut.

It is known by the work of Leighton and Rao [44] that (approximate) solutions to the SPARSEST CUT problem can be used to compute approximations to EDGE SEPARATOR. They also show

that the solutions to the latter problem can subsequently be used to approximate the BISECTION problem. Additionally Simon and Teng [65] build on this work and use the solutions to EDGE SEPARATOR in order to approximate the k-BALANCED PARTITIONING problem. These approximation techniques can be used for any graph class and were developed since the problems are NP-hard [31] in general. We will use these techniques on solid grid graphs in Section 3.3 by showing how solutions to the SPARSEST CUT problem can be computed for these graphs. In particular we show that constant approximations to SPARSEST CUT can be computed in $\mathcal{O}(n)$ time. This improves on the previously fastest known algorithm by Park and Phillips [55]. Their algorithm however is more general and will compute a constant approximation for any planar graph. In particular they show how to compute an $\mathcal{O}(t)$ approximation to the SPARSEST CUT problem in $\mathcal{O}(n^{1+1/t} \log^3 n)$ time for any planar graph.

Using the above methods we are able to compute solutions to the BISECTION problem which approximate both the cut size and the sizes of the cut out parts, in the following sense.

Definition 3.1. Let a partition of the n vertices of a graph into k parts be given. It is said to be *near-balanced* if there is an $\varepsilon > 0$ such that each part has size at most $(1 + \varepsilon)\lceil n/k \rceil$. If each set has size at most $\lceil n/k \rceil$ we call the partition *perfectly balanced*.

The presented algorithm is able to compute near-balanced solutions for any $\varepsilon > 0$. It approximates the cut size within a factor of $\alpha \in \mathcal{O}(1/\varepsilon^3)$, and the runtime is $\mathcal{O}(n^{1.5})$ if ε is constant. For the k-BALANCED PARTITIONING problem we obtain an algorithm that computes a partition into k sets that deviates by a factor of 2 from being perfectly balanced. That is, each set has size at most $2\lceil n/k \rceil$. The cut size of the solution approximates the optimal perfectly balanced partition by a ratio of $\alpha \in \mathcal{O}(\log k)$. On solid grid graphs the runtime of the algorithm is $\mathcal{O}(n^{1.5} \log k)$. Both of these algorithms improve the runtime of the fastest known ones for these problems on solid grid graphs, due to our improvement over the algorithm by Park and Phillips [55].

Since the techniques used in the two parts of this chapter differ considerably, an overview is given separately in each respective section.

3.1.1 Related Work

The related work on computing optimal solutions to the BISECTION problem can be found in Section 2.1.2 (page 14). Here we only survey the related work for approximating BISECTION. For perfectly balanced solutions and general graphs, assuming the Unique Games Conjecture no constant approximations can be computed in polynomial time [38]. The best approximation algorithm known [57] achieves a ratio of $\mathcal{O}(\log n)$ on the cut size. If the given graph is *dense*, i.e. its minimum degree is $\Omega(n)$, Arora *et al.* [3] give a PTAS. Also for planar graphs Díaz *et al.* [16] show how to achieve a PTAS w.r.t. the cut size. When near-balanced solutions are allowed, as mentioned above, Leighton and Rao [44] give a technique to compute such solutions. The cut size is approximated within $\alpha \in \mathcal{O}(\beta/\varepsilon^3)$ where β is the approximation ratio of computing a solution to SPARSEST CUT.

The SPARSEST CUT problem is NP-hard in general [51] but can be approximated [4] within a ratio of $\mathcal{O}(\sqrt{\log n})$. For planar graphs it can however be computed optimally in polynomial time using the algorithm by Park and Phillips [55]. They devise a method which assigns positive and negative weights to edges in the directed version of the dual graph. This is done in a way such that any simple (directed) cycle in this graph has a total absolute weight that equals the number of vertices of the primal graph that lie inside of the cycle. A crucial observation the authors use is that in a planar graph there is an optimal solution to SPARSEST CUT that contains exactly one segment. Hence it corresponds to a simple cycle in the directed dual graph. The cycle with the least number of edges having a fixed total weight can be computed using a dynamic program. Therefore the cut with the smallest ratio between number of edges and cut out vertices can be found in polynomial time. In the same paper Park and Phillips also show how the SPARSEST CUT problem can be approximated within a factor of $\mathcal{O}(t)$ in $\mathcal{O}(n^{1+1/t} \log^3 n)$ time

for these graphs, building on the above insights.

Also the EDGE SEPARATOR problem is NP-hard [70] in general. In fact it is NP-hard for any $b = \frac{c}{n^{1-d}}$ where $c, d > 0$ are arbitrary constants. Using the method of Arora *et al.* [4] it can be approximated within a factor of $\mathcal{O}(\sqrt{\log n})$, where however also the sizes of the resulting cut out parts are approximated within a constant factor. If the given input graph is dense then there is a PTAS for the problem [3].

We defer the related results on the k-BALANCED PARTITIONING problem to the next chapter (Section 4.1.2). We only note at this point that for graphs with excluded minors (such as planar graphs) it is possible to apply a spreading metrics relaxation [20] and the famous Klein-Plotkin-Rao Theorem [40] to compute solutions with a constant approximation factor on the cut size. The set sizes will be at most $2\lceil n/k \rceil$. However this algorithm needs $\widetilde{\mathcal{O}}(n^3)$ time (ignoring logarithmic factors), or $\widetilde{\mathcal{O}}(n^2)$ expected time. Hence it is slower than our corresponding deterministic algorithm for solid grid graphs, even though it gives a better ratio on the cut size.

3.2 Corner Cuts are Close to Optimal

In this section we will show that optimal corner cuts are good approximations to optimal m-cuts. More formally, the main result of this section is summarised in the following theorem.

Theorem 3.2. *Let C^* be the cut size of an optimal m-cut in a solid grid graph G and $\varepsilon \in]0, 1]$. Then there exists a corner m'-cut, for some $m' \in [(1-\varepsilon)m, (1+\varepsilon)m]$, which has a cut size that is at most a factor of $\mathcal{O}(1/\sqrt{\varepsilon})$ larger than C^*.*

We will prove Theorem 3.2 by going through several steps, of which each solves an interesting problem of its own. We start by comparing cuts in grid graphs to cuts in polygons in order to be able to use the continuous nature of the polygons in our proofs. For this we convert a given solid grid graph into a simple orthogonal polygon, and hence all polygons considered in this

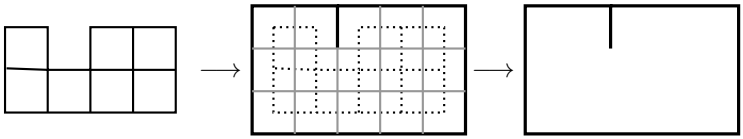


Figure 3.3: *Converting a solid grid graph to a polygon.*

section are orthogonal and simple. We define a polygon using its interior point set.

Definition 3.3 (polygon)**.** A *polygon* $\mathscr{P} \subset \mathbb{R}^2$ is an open bounded set of points in the plane. Let β be the boundary of $\mathscr{P}$. If β only contains axis-parallel line segments we refer to $\mathscr{P}$ as *orthogonal*. We call $\mathscr{P}$ *simple* if any closed curve in $\mathscr{P}$ can be shrunk to a point without leaving $\mathscr{P}$.

Given a solid grid graph $G = (V, E)$, consider its natural embedding in the plane where each vertex is a coordinate in $\mathbb{N}^2$. The conversion is done by replacing each vertex $(x, y) \in V$ by a unit square that has its centre at the coordinate (x, y) (Figure 3.3). Notice that the squares of two neighbouring vertices of V will share a boundary, but the converse is not necessarily true. Ignoring those boundaries that correspond to an edge in G leaves a connected curve that is the boundary of the polygon. It may happen that this boundary is degenerate in the sense that it can have overlapping edges (Figure 3.3). The region enclosed by the boundary is the polygon $\mathscr{P}_G$ and has area exactly n, equal to the number of vertices in G.

All the notions used for cuts in grids carry over naturally to the case of polygons. Intuitively, the building blocks of a cut in a polygon $\mathscr{P}$ are curves that can be drawn between points on the boundary of $\mathscr{P}$ (Figure 3.4). In accordance with the grid case we call them *segment curves* and a *cut* is a set of segment curves. Formally these curves are defined as follows.

Definition 3.4 (curve, boundary point, segment curve)**.** Given a polygon $\mathscr{P}$ a *curve* $\lambda \subset \mathscr{P}$ is the image of a continuous map from the unit interval to $\mathscr{P}$. The *length* of a curve is measured using the l_1-norm. If not otherwise stated, all considered curves have finite length. If β denotes the boundary of $\mathscr{P}$, we call a

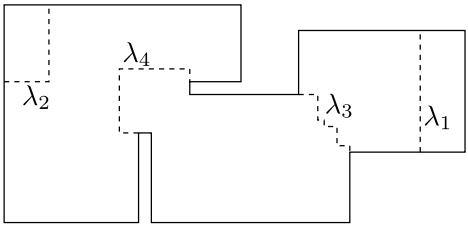


Figure 3.4: *A straight, corner, staircase, and rectangular line in a polygon denoted by λ_1 through λ_4 respectively.*

point $p \in \beta$ a *boundary point* of a curve λ in $\mathscr{P}$ if the distance from p to λ is 0. If λ has two boundary points we call it a *segment curve*.

Note that a segment curve has exactly two boundary points since a polygon is an open set of points. Consider the connected areas left after removing the segment curves from a polygon. An *m-cut* is a set L of such curves that leaves a subset of these areas with total size m. The *cut size* of L is the sum of the lengths of the curves in L, which are measured using the Manhattan distance. This ensures that an m-cut in a grid graph G has a corresponding m-cut in the polygon $\mathscr{P}_G$ with the same cut size. The curves in the latter cut reside on the boundaries of the unit squares used to construct $\mathscr{P}_G$. Note that the m-cut in $\mathscr{P}_G$ that corresponds to the optimal m-cut in G obviously has a cut size that is at least the cut size of the *optimal* m-cut in $\mathscr{P}_G$. The latter is defined as an m-cut having the smallest cut size among all m-cuts. Those segment curves that we will use to cut out areas from polygons are rectilinear and we therefore call them lines. A *corner m-cut* in a polygon is an m-cut containing only *straight* and *corner lines*. Analogous to the case of grids, the former are orthogonal segment curves without bends, and the latter are orthogonal and have exactly one right-angled bend, as seen in the following definition.

Definition 3.5 (bar, straight, corner line)**.** We call a curve λ a *bar line* if all points in λ share either the same x- or the same y-coordinate. In the former case we say that the orientation of the bar line is *vertical* and it is *horizontal* in the latter case. A

bar line that is also a segment curve is called a *straight line*, and a segment curve that consists of a horizontal and a vertical bar line is called a *corner line*. We refer to these bar lines as the horizontal, respectively vertical, *bar line of* the straight or corner line. Analogous to the corner segments, we call the point at which the two bar lines of a corner line meet its *corner*, and say that it *points* in two of the directions *up*, *down*, *left*, and *right*, depending on whether its horizontal and vertical bar lines go up, down, left, or right from its corner, respectively.

3.2.1 An Overview of the Used Techniques

We will first show the existence of corner cuts in simple polygons that cut out almost the desired area and have small cut size (close to optimal). We will then convert such a cut in a polygon $\mathscr{P}_G$ derived from a grid graph G to a corresponding cut in G having the properties described in Theorem 3.2. More precisely we prove the following results for polygons which together imply the theorem.

1. We show that there is an optimal m-cut in a polygon that is almost a corner cut, in the sense that the cut consists of only straight and corner lines except at most one other segment curve. This curve may be shaped like a staircase (a so called *staircase line*), or it may be a *rectangular line*, which is defined as a continuous part of the boundary of an orthogonal rectangle (Figure 3.4).

2. We show how to remove a rectangular line from a cut containing only straight and corner lines otherwise. We replace the rectangular line by a set of straight and corner lines, and at most one staircase line. Together these cut out the same area as the rectangular line. While doing this we need to take other curves from the cut into consideration so that the newly introduced curves do not interfere with these. The new cut will also be an m-cut but its cut size may not be optimal. However, we show that the cut size of the new cut is only a constant factor away from the optimal.

3. Given an m-cut of the polygon consisting of straight and corner lines, and one staircase line, we next show how to replace the staircase line with a set of corner and straight lines, such that the new area that is cut out is close to m. To be more precise, the new cut is an m'-cut where $m' \in [(1-\varepsilon)m, (1+\varepsilon)m]$ for any desired value $\varepsilon \in]0,1]$. Further, the cut size of the new cut is only a factor $\mathcal{O}(1/\sqrt{\varepsilon})$ times the cut size of the original cut.

4. Finally we show how to convert a cut containing only straight and corner lines in a polygon $\mathscr{P}_G$ corresponding to a grid graph G into a cut in G. Note that this step would be straightforward if all the curves in the cut were passing through exactly the midpoints of the edges of the grid. We call such curves *grid lines*. We show that all curves in the cut obtained in the previous steps can be moved to grid lines in such a way that the cut size remains the same, but we lose a small area a from the cut out area. Since a is small we can cut this area from the polygon using a recursive method using only grid lines so that the cut size grows by only a small factor.

The following sections explain these techniques in more detail.

3.2.2 Cuts in Polygons

We will now show that in an optimal m-cut of a polygon all but at most one curve are corner and straight lines. Curves with more bends include *staircase lines* and *rectangular lines*. The former have at least two bends and are monotonic in x- and in y-direction. The latter have two or three bends and form part of the boundary of an orthogonal rectangle (Figure 3.4).

Definition 3.6 (staircase line)**.** For any polygon $\mathscr{P}$ a *staircase line* $\lambda \subseteq \mathscr{P}$ is a segment curve that consists of a sequence of bar lines such that of two adjacent bar lines one is horizontal and the other vertical. This sequence has length at least three and the resulting curve is monotonic in x- and y-direction. The orientation of λ is *up* if its left boundary point also is lower than the other, and *down* otherwise.

Definition 3.7 (rectangular line)**.** Let $\mathscr{R} \subset \mathbb{R}^2$ be an axis parallel rectangle in the plane and let γ be its boundary. Any segment curve $\lambda \subseteq \gamma \cap \mathscr{P}$ which contains either two or three corners of $\mathscr{R}$ is called a *rectangular line*. These corners are called the *corners of* λ. We call $\mathscr{R}$ the *defining rectangle of* λ if $\mathscr{R}$ is the rectangle of smallest size among those from which λ can be constructed in this way.

Notice that a rectangular line contains either three or four bar lines between its corners and boundary points since $\mathscr{P}$ is an open set. Notice also that a corner line and a staircase line that have the same boundary points have the same length.

In a first step, we convince ourselves that in any simple polygon there is an optimal m-cut that contains only straight, corner, staircase, and rectangular lines. Furthermore, none of these lines *cross* or *overlap*, which is defined as follows.

Definition 3.8 ($\mathscr{A}$- and $\mathscr{B}$-part, crossing, overlapping)**.** Let $\mathscr{A}(L) \subseteq \mathscr{P} \setminus \{p \in \lambda \mid \lambda \in L\}$ be the open set of size m that is cut out by the m-cut L in $\mathscr{P}$ and let $\mathscr{B}(L) = \mathscr{P} \setminus (\mathscr{A}(L) \cup \{p \in \lambda \mid \lambda \in L\})$ be the other cut out open set of size $n - m$. That is, the areas $\mathscr{A}(L)$ and $\mathscr{B}(L)$ do not include points that are contained in curves of L or the boundary of $\mathscr{P}$.

Let λ_1 and λ_2 be two segment curves in $\mathscr{P}$. We say that λ_1 and λ_2 *cross* if λ_2 contains points from both $\mathscr{A}(\{\lambda_1\})$ and $\mathscr{B}(\{\lambda_1\})$. A cut L is said to be *non-crossing* if no pair of curves in L cross. Two segment curves in $\mathscr{P}$ *overlap* if they do not cross but share a curve of length greater than zero (i.e. the shared part is not just a point).

The following results are analogous to those obtained in [54] for grid graphs.

Lemma 3.9. *In any polygon $\mathscr{P}$ there is an optimal m-cut L that is non-crossing and contains only straight, corner, staircase, and rectangular lines. Furthermore no curves in L overlap.*

Proof. Note that any pair of crossing segment curves can be seen as a (different) pair of segment curves that do not cross. Hence

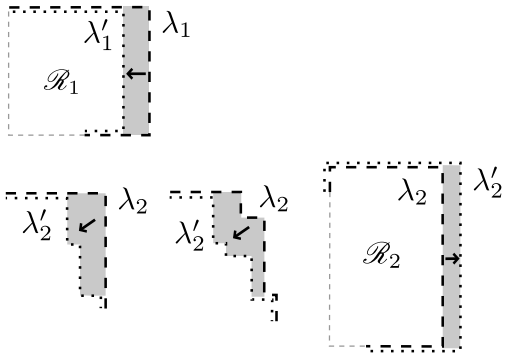


Figure 3.5: *A rectangular line λ_1 with its defining rectangle $\mathscr{R}_1$ is replaced with the rectangular line λ_1' (top). To compensate for the area a (shaded in grey), another curve λ_2 is replaced by λ_2' (bottom). It can be a corner, staircase, or a rectangular line (with defining rectangle $\mathscr{R}_2$).*

there always exists an optimal non-crossing m-cut. Additionally, removing overlapping parts of curves results in an m-cut of smaller cut size and thus no curves in L overlap. Also, as in the case of grids, it is easy to see that for any m-cut with cut size C there is an m-cut for which every curve is a segment curve and has a cut size of at most C. Thus, let λ be a curve from L and let $\mathscr{R} \subset \mathbb{R}^2$ be the smallest rectangle containing λ. Due to the well-known isoperimetric problem, using the l_1-norm (see e.g. [67]) it follows that λ is a rectangular line if the boundary points of λ do not coincide with two of the opposing corners of $\mathscr{R}$. If λ's boundary points coincide with two opposing corners of $\mathscr{R}$ it is easy to see that λ can be replaced with a straight, corner, or staircase line since these lines have minimum length between the boundary points using the l_1-norm. □

In a next step, we show that if an optimal m-cut contains a rectangular line, then all other curves are straight or corner lines. Generally speaking, the reason is that cuts can be modified so that the cut out area remains the same. This is easy to see for two rectangular lines where the $\mathscr{A}$-part of the cut out area is on

the inside of one of the rectangles and on the outside of the other: we can simply make both rectangular lines smaller by the same area, thereby decreasing the length of the cut (Figure 3.5)—a contradiction to optimality. More generally, we call a corner line *convex* w.r.t. the area next to its 90 degree angle and *concave* w.r.t. the area next to its 270 degree angle (Figure 3.6). Similarly, a rectangular line is convex w.r.t. the area next to its 90 degree angles, and concave w.r.t. the area on its other side. Similar area exchange arguments show that for an optimal m-cut with a rectangular line, the area on its concave side will belong to the same part of the cut as the area on the concave sides of all corner lines. This fact will become important later when a rectangular line is replaced with a staircase line.

Definition 3.10 (convex, concave)**.** For any segment curve $\lambda \in L$ let $\mathscr{C} \subseteq \mathscr{P}$ be an open set of points such that λ is part of the boundary of $\mathscr{C}$. We define $\mathscr{X}(\mathscr{C}) \subseteq \mathscr{C}$ as the set of points $p \in \mathscr{C}$ such that there exist a horizontal and a vertical bar line which both are contained in $\mathscr{C}$, and end in p and a point on λ. We call a corner or rectangular line *convex* w.r.t. $\mathscr{C}$ if $\mathscr{X}(\mathscr{C}) \neq \emptyset$ and *concave* w.r.t. $\mathscr{C}$ otherwise.

Since in an optimal m-cut L no curves overlap, each set $\mathscr{A}(L)$ and $\mathscr{B}(L)$ can only lie to one side of a corner or rectangular line. Hence any such line in L is either concave or convex w.r.t. one of the cut out areas from $\mathscr{P}$.

For staircase lines, area exchange works by changing the staircase line while still keeping it monotonic between its end points. The potential area exchanged is the *deficit* or the *surplus*, which are areas with monotone boundaries contained in the $\mathscr{B}$- and $\mathscr{A}$-part respectively (Figure 3.7). These areas are used to prove that an optimal cut requires at most one staircase line: for more than one staircase line we trade the smaller deficit or surplus of one staircase with the larger of another one, turning the former into only straight and corner lines (Figure 3.8).

More formally, consider the simple case of an m-cut that contains only one curve λ which is a staircase line. In this case the set $\mathscr{X}(\mathscr{C}) \cup \lambda$ is referred to as the surplus if $\mathscr{C} = \mathscr{A}(\{\lambda\})$ and as the deficit if $\mathscr{C} = \mathscr{B}(\{\lambda\})$. We are interested in the segment

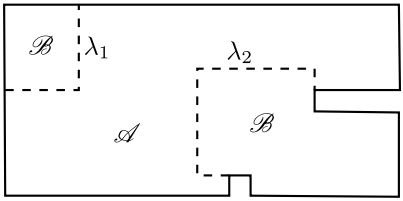


Figure 3.6: *A corner, and rectangular line in a polygon denoted by λ_1, λ_2, respectively. Both are concave with respect to the $\mathscr{A}$-part and convex with respect to the $\mathscr{B}$-part.*

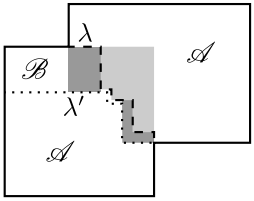


Figure 3.7: *A staircase line λ together with its surplus (in light grey shading) and its deficit (in dark grey shading).*

curves that are part of the boundary of the deficit and surplus. We need to add the points in λ to $\mathscr{Z}(\mathscr{C})$ so that the boundary of both the deficit and the surplus is made up of segment curves. (For instance the boundary of the surplus shown in Figure 3.7 would otherwise only contain λ as a segment curve.) If λ is the only curve in an m-cut then the segment curves apart from λ in the surplus and deficit are all straight and corner lines by definition of $\mathscr{Z}(\mathscr{C})$. The surplus can be seen as the area that λ cuts out from the $\mathscr{A}$-part in addition to what these lines in the boundary of the surplus cut out. The deficit on the other hand can be seen as the area that λ does not cut out compared to the lines in the boundary of the deficit. If there are other curves apart from λ in an m-cut then the definition has to be modified in the following way, in order to capture a similar notion. If there is a curve λ' that overlaps with λ (as shown in Figure 3.7) then it can happen that the intersection between $\mathscr{Z}(\mathscr{C})$ and a part cut out by λ' is non-empty. This would mean that a curve in $\mathscr{Z}(\mathscr{C}) \cup \lambda$ might cross λ'. Such a curve will later be used when transforming λ. Hence all the parts cut out by other curves that include λ are removed in the surplus and deficit.

Definition 3.11 (surplus, deficit)**.** Let $\lambda \in L$ be a staircase line from a non-crossing m-cut L, $\mathscr{C} \in \{\mathscr{A}(L), \mathscr{B}(L)\}$, and $L' = L \setminus \{\lambda\}$. For any curve $\lambda' \in L'$ let $\mathscr{D}_{\lambda'} \in \{\mathscr{A}(\{\lambda'\}), \mathscr{B}(\{\lambda'\})\}$

such that $\lambda \cap \mathscr{D}_{\lambda'} = \emptyset$. We call the set

$$(\mathscr{Z}(\mathscr{C}) \cup \lambda) \setminus \bigcup_{\lambda' \in L'} \mathscr{D}_{\lambda'}$$

the *surplus* of λ if $\mathscr{C} = \mathscr{A}(L)$ and we call it the *deficit* of λ if $\mathscr{C} = \mathscr{B}(L)$.

Using the above notions we are able to prove that if there is a rectangular line in an optimal m-cut it is the only curve that is not a corner or straight line. We proceed in two steps of which the following lemma is the first.

Lemma 3.12. *For any polygon $\mathscr{P}$, if an optimal m-cut L contains a rectangular line that is concave with respect to the area $\mathscr{C} \in \{\mathscr{A}(L), \mathscr{B}(L)\}$ then it contains no staircase line and also no corner line that is convex with respect to $\mathscr{C}$.*

Proof. Let $\lambda_1 \in L$ be the rectangular line that w.l.o.g. is concave with respect to $\mathscr{C} = \mathscr{A}(L)$ (by Lemma 3.9 no curves overlap and hence any rectangular or corner line is either concave or convex w.r.t. $\mathscr{C}$). Let $\lambda_2 \in L$ be a staircase line. As we will show, there is a sufficiently small area of size $a > 0$ which can be locally "transferred" from λ_1 to λ_2 by making λ_1 shorter while transforming λ_2 such that the cut size is decreasing. Hence we get a contradiction to the optimality of L.

Any rectangular line has at least two adjacent corners, i.e. there is a bar line connecting them. For any rectangular line under consideration we can assume w.l.o.g. that these corners coincide with the lower right and upper right corners of its defining rectangle. Let $\mathscr{R}_1$ be the defining rectangle of λ_1, let $\mathscr{Q}_1(x) = \{(x', y') \in \mathscr{R}_1 \mid x' > x\}$, and let $a_1(x)$ be the size of $\mathscr{Q}_1(x)$. For sufficiently small $a > 0$ there is a value x_a such that $a_1(x_a) = a$ and the rectangle $\mathscr{R}_1 \setminus \mathscr{Q}_1(x_a)$ defines a rectangular line λ_1' which has the same boundary points as λ_1 and does not cross any curve in L. Observe that λ_1' is shorter than λ_1 by twice the width of the area $\mathscr{Q}_1(x_a)$. When replacing λ_1 with λ_1' in L we need to compensate for the area $\mathscr{Q}_1(x_a)$ in order to cut out an area of size m, by also replacing the staircase line λ_2

with some appropriate curve λ_2' (see Figure 3.5). We show next how this is done.

Since λ_2 is a staircase line, for sufficiently small a we can find a staircase line λ_2' that cuts out an area of size a from the surplus of λ_2 (remember that $\mathscr{C} = \mathscr{A}(L)$), such that λ_2 and λ_2' have the same boundary points, and replacing λ_2 with λ_2' will make the cut out area have size m (remember that λ_1 is concave w.r.t. $\mathscr{C}$). Notice that, by the definition of the surplus, λ_2' does not cross any curve and therefore the new m-cut is non-crossing. The length of λ_2' is equal to the length of λ_2 in the l_1-norm and hence the cut size is decreasing when replacing λ_1 and λ_2. This is a contradiction to the optimality of L and therefore λ_2 cannot be a staircase line.

A similar argument can be made when λ_2 is a corner line that is concave w.r.t. $\mathscr{C}$. For sufficiently small a we can find a staircase line λ_2' which has a deficit of size a in the m-cut that results from replacing λ_1 and λ_2, and the boundary of the deficit is $\lambda_2 \cup \lambda_2'$. Also if a is small enough, λ_2' does not cross any other curve since no curve overlaps with λ_2 by Lemma 3.9. Since this means that the boundary points of λ_2 and λ_2' are the same, the length of these two lines are the same in the l_1-norm, and therefore the cut size decreases when replacing λ_1 and λ_2. This is a contradiction to the optimality of L and hence λ_2 cannot be a corner line that is concave w.r.t. $\mathscr{C}$. □

Using the above lemma we can prove that if an optimal m-cut contains a rectangular line then no other curve is a rectangular or staircase line, as the following lemma shows.

Lemma 3.13. *For any polygon $\mathscr{P}$, if an optimal m-cut L contains a rectangular line that is concave with respect to the area $\mathscr{C} \in \{\mathscr{A}(L), \mathscr{B}(L)\}$ then all other curves are straight and corner lines, where the latter all are concave with respect to $\mathscr{C}$.*

Proof. By Lemma 3.9 we can assume that all curves in L are straight, corner, staircase, or rectangular lines. Additionally Lemma 3.12 shows that apart from the rectangular line $\lambda_1 \in L$ there can only be rectangular lines that do not fit the statement

in this lemma. Let $\lambda_2 \in L$ be such a rectangular line, and we first consider the case when λ_2 is convex w.r.t. $\mathscr{C}$. As in the proof of Lemma 3.12 let λ_1' be the rectangular line that is defined by the rectangle $\mathscr{R}_1 \setminus \mathscr{Q}_1(x_a)$. Analogous to the definition of λ_1' we can define a rectangular line λ_2' such that the corresponding function $a_2(\cdot)$ equals a for an appropriate value x_a' if a is sufficiently small. The line λ_2' is shorter than λ_2 by twice the width of the corresponding area $\mathscr{Q}_2(x_a')$. But this means that replacing λ_1 with λ_1' and λ_2 with λ_2' results in an m-cut with smaller cut size than L. This contradicts the optimality of L and hence λ_2 cannot be a rectangular line that is convex w.r.t. $\mathscr{C}$.

Thus consider the case when λ_2 is a rectangular line that also is concave w.r.t. $\mathscr{C}$ (see Figure 3.5). For $i \in \{1, 2\}$ let h_i and w_i be the height and width of the defining rectangle $\mathscr{R}_i$ of λ_i, respectively. Assume w.l.o.g. that $w_2 \geq h_2 \geq h_1$ (otherwise we can switch the identity of the width and height of $\mathscr{R}_2$ for the former, and the identity of λ_1 and λ_2 for the latter inequality). As noted before, the length l_1' of λ_1' is shorter than the length l_1 of λ_1 by twice the width of $\mathscr{Q}_1(x_a)$. Since the height of the latter equals the height of $\mathscr{R}_1$ this means that $l_1' = l_1 - 2a/h_1$. If λ_2 has three corners then let (x, y) be the corner that is adjacent to both the other two corners. In case λ_2 has two corners we can decompose it into three bar lines of which two are incident to exactly one corner. Let in this case (x, y) be the corner that is incident to the longer of these two bar lines. In all of these cases we can assume w.l.o.g. that (x, y) is the top right corner of $\mathscr{R}_2$. For sufficiently small a we can find a rectangular line λ_2' that has the following properties. It is defined by a rectangle $\mathscr{R}_2'$ that has the same lower left corner as $\mathscr{R}_2$ and the top right corner $(x + z, y + z)$, for some $z > 0$, such that the area $(\mathscr{R}_2' \setminus \mathscr{R}_2) \cap \mathscr{P}$ that is cut out between λ_2' and λ_2 has size a. It also does not cross any other curve, and λ_2' shares at least one boundary point p with λ_2. By the assumption that (x, y) is the top right corner of $\mathscr{R}_2$ and the construction of $\mathscr{R}_2'$, the boundary point p is the one that is incident to the lower horizontal bar lines of λ_2 and λ_2'. Since by Lemma 3.9 no curves overlap in L, such that for sufficiently small a the constructed line does not cross any other curve, this means that λ_2' always exists. Notice however that the two lines might differ in the other boundary point if λ_2

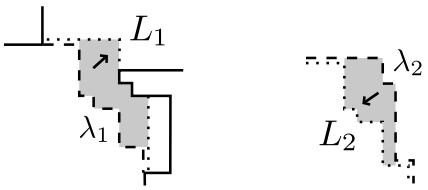


Figure 3.8: *Two staircase lines λ_1 and λ_2 together with their respective surplus (or deficit) shaded in grey. The dotted lines indicate that the boundary of the surplus (or deficit) together with some parts of λ_i, $i \in \{1, 2\}$, form only corner and straight lines which are contained in L_i.*

has two corners since the boundary of $\mathscr{P}$ may overlap with the boundary of $\mathscr{R}'_2$. Under the assumption that $\mathscr{P}$ is orthogonal we can always find some sufficiently small $z > 0$ such that the area $\mathscr{R}'_2 \setminus \mathscr{R}_2$ is entirely included in $\mathscr{P}$ though.

The area $\mathscr{R}'_2 \setminus \mathscr{R}_2$ can be decomposed into three rectangles of which one extends $\mathscr{R}_2$ to the right by z, one extends $\mathscr{R}_2$ to the top by z, and one which lies between these two extensions and has height and width z. By the assumption that $w_2 \geq h_2$ we can therefore conclude that $a = zw_2 + zh_2 + z^2 > 2zh_2$. It is easy to see that the length l'_2 of λ'_2 is at most $l_2 + 4z$. Solving the lower bound on a for z we can conclude that $l'_2 < l_2 + 4\frac{a}{2h_2} = l_2 + 2a/h_2$. Replacing λ_1 and λ_2 by λ'_1 and λ'_2 yields an m-cut that has a shorter cut size than L since we assumed that $h_2 \geq h_1$. But this contradicts the optimality of L which means that λ_2 cannot be a rectangular line that is concave w.r.t. $\mathscr{C}$. □

After considering optimal m-cuts containing rectangular lines we turn to the case where they contain staircase lines. In this case we can show that there always exists an optimal m-cut in which at most one curve is a staircase line while all others are corner and straight lines.

Lemma 3.14. *For any polygon $\mathscr{P}$, if there is an optimal m-cut L that contains a staircase line then there also is an optimal m-cut that contains at most one staircase line while all other curves are straight or corner lines.*

Proof. By Lemma 3.13 it can not happen that there is a rectangular line in L. Hence, by Lemma 3.9, the only case we have to consider is when there are two staircase lines λ_1 and λ_2 in L. It can happen that the boundary of the deficit of λ_2 contains parts of λ_1, or that the boundary of the surplus of λ_1 contains parts of λ_2. It is easy to see though that it can not happen that both boundaries contain parts of the respective other staircase line. Hence we can assume w.l.o.g. that the boundary of the surplus of λ_1 does not contain any parts of λ_2. Let a_1 denote the size of the surplus of λ_1. For any $a \in [0, a_1]$ we can find a set of curves $L_1(a)$ that cut out an area of size a from the surplus of λ_1, such that removing λ_1 yields a $(m-a)$-cut. If $a < a_1$ we can choose a single staircase line having the same boundary points as λ_1 for the set $L_1(a)$. If $a = a_1$ the curves in $L_1(a)$ are part of the boundary of λ_1's surplus together with some parts of λ_1 (Figure 3.8).

If the boundary of the deficit of λ_2 contains parts of λ_1, the deficit of λ_2 can grow when replacing λ_1 with $L_1(a)$. Hence let $d_2(a)$ denote the size of λ_2's deficit in the constructed $(m-a)$-cut. Similar as for the surplus of λ_1, for a fixed a we can find a set of curves $L_2(d)$ for any $d \in [0, d_2(a)]$ cutting out an area of size d from the deficit of λ_2. It either contains a single staircase line or curves that are part of the boundary of λ_2's deficit. Let $b = \min\{a_1, d_2(a_1)\}$. Observe that it is possible to replace the line λ_2 with the set $L_2(b)$ after replacing λ_1 with $L_1(b)$. This yields a $(m-b+b)$-cut, i.e. an m-cut which has a cut size that is at most the cut size of the original m-cut since distances are measured using the l_1-norm. Assume that some curve in $L_1(b)$ or in $L_2(b)$ overlaps with some other curve in the m-cut (including those in $L_2(b)$ and $L_1(b)$ respectively). The overlapping part can be removed, again yielding an m-cut which now however has a shorter cut size. Since this is a contradiction to the optimality of L, we can conclude that no curve in $L_1(b)$ or $L_2(b)$ overlaps with any other curve. By the definition of b, at least one (both if $a_1 = d_2(a_1)$) of the sets $L_1(b)$ and $L_2(b)$ consists of curves that are part of the boundary of the surplus or deficit, respectively, together with some parts of the respective staircase line. Since these curves do not overlap with any other curves they must either be straight or corner lines, by the definition of the surplus

and deficit (cf. Figure 3.8).

Hence if L contains several staircase lines then there is an optimal m-cut which contains one staircase line less. By applying the argument repeatedly we can conclude that there is an optimal m-cut with at most one staircase line while all other curves are straight or corner lines. □

To summarise the above results, the following observation immediately follows from Lemmas 3.9, 3.13, and 3.14.

Corollary 3.15. *For any simple polygon $\mathscr{P}$ there is an optimal m-cut L such that all curves in L are corner or straight lines except at most one which is either a staircase line or a rectangular line. If there is a rectangular line in L that is concave with respect to the cut out area $\mathscr{C} \in \{\mathscr{A}(L), \mathscr{B}(L)\}$ then all corner lines in L are concave with respect to the same area $\mathscr{C}$.*

Because our interest is in cuts with only straight and corner lines, we need to study how we can cope with a rectangular line, and how with a staircase line. For a rectangular line we show how to convert the m-cut into a cut in which there is at most one staircase line while the cut size grows at most by some constant factor. With our observations on staircase lines we are then able to convert any optimal m-cut into one containing only straight and corner lines.

3.2.3 Removing Rectangular Lines

We now show how to convert an optimal m-cut containing straight and corner lines and one rectangular line into an m-cut containing only straight and corner lines except at most one which is a staircase line. Consider the area inside the defining rectangle of the rectangular line (Figure 3.9). This region may contain a part of the boundary of the polygon (and possibly some other curves of the cut). We can replace the rectangular line with a set Ξ of straight and corner lines lying within the defining rectangle such that these curves have total length less than the length of the rectangular line. By doing this, we do

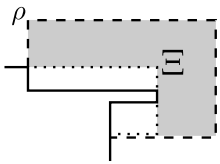


Figure 3.9: *A rectangular line ρ (dashed) together with the set of curves Ξ (dotted) with which it is replaced. The area of size a is shaded in grey.*

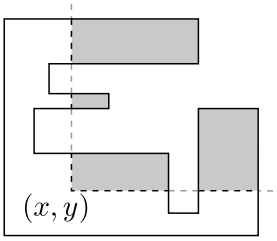


Figure 3.10: *A virtual corner line (black dashed) at (x, y). The cut out area is shaded in grey.*

not increase the size of our cut, but we now have to cut out an additional area of size a equal to the difference in sizes of the part cut out by the original cut and the part cut out by the new cut. We show how to find a set of curves that cut out the required area of size a and has total length not too large (compared to the cut size C of the optimal m-cut). Note that the length of the rectangular line (and thus C) is at least $\sqrt{a}$. So, it is sufficient to show that the area of size a can be cut out using a set of curves of total length not much larger than $\sqrt{a}$.

In order to find this set of curves we need to abstract from the actual topology of the polygon. We achieve this by introducing the following notions (cf. Figure 3.10). In the proofs of this section we will restrict ourselves to the case of one specific orientation of the involved curves. Notice that in Definition 3.4 a segment curve can be defined for the plane by seeing $\mathbb{R}^2$ as the polygon with a boundary that lies infinitely far away. Hence we may define corner lines of infinite length in the plane due to Definition 3.5, and leverage the following definition.

Definition 3.16 (virtual corner line)**.** Let μ be a corner line in the plane $\mathbb{R}^2$. For any (open or closed) finite area $\mathscr{P} \subset \mathbb{R}^2$ the set Λ of corner and straight lines in $\mathscr{P}$ for which $\lambda \in \Lambda$ if and only if $\lambda \subset \mu \cap \mathscr{P}$ is called a *virtual corner line*. The corner of μ is also referred to as the *corner of* Λ. The *length* of Λ is the sum of the lengths of the included straight and corner lines.

The *horizontal length* of Λ is the total length of all horizontal bar lines covered by Λ, while the *vertical length* of Λ is the total length of all vertical bar lines covered by Λ. If Λ cuts out an area of size a on the upper right side of its corner, we say that it is a virtual corner line *for* a.

For a fixed value a let $\Lambda(x)$, if it exists, be a virtual corner line for a with corner (x, y) such that its underlying corner line in the plane points up and right. If there are several virtual corner lines that match the definition then $\Lambda(x)$ denotes the one having the largest y value for its corner. Let $l_h(x)$ be the horizontal, $l_v(x)$ the vertical, and $l(x) = l_h(x) + l_v(x)$ the total length of $\Lambda(x)$. Also let $\mathscr{P}(x) \subset \mathscr{P}$ be the cut out area of size a, i.e. $\Lambda(x)$ is the lower and left boundary of $\mathscr{P}(x)$.

Notice that if $\mathscr{P}$ has size n, for any $a \in [0, n]$ there is a value x' such that $\Lambda(x) = \Lambda(x')$ for all $x \leq x'$ while $\Lambda(x) \neq \Lambda(x')$ for all $x > x'$. Also there is a value x'' such that $\Lambda(x)$ is defined for all $x \leq x''$ while $\Lambda(x)$ is not defined whenever $x > x''$. In this sense the points x' and x'' are extreme points for these virtual corner lines beyond which the function $\Lambda(x)$ is irrelevant for our purposes. Let $I_a = [x', x'']$ be the interval of relevant x values for the virtual corner lines for a. Note that the y values of the corners of these virtual corner lines are non-increasing with x in I_a.

The easy case is when the required area a can be cut out from the polygon using a single virtual corner line of short length (say, of length at most $c\sqrt{a}$ for some fixed constant c). However, depending on the shape of the polygon, it is not always possible to find such a virtual corner line. For example, in the polygon shown in Figure 3.11, any virtual corner line cutting out the required area has a long vertical or a long horizontal length.

Given any polygon we can search along the x-axis between the two extremities of the polygon, and for each value of x find a y such that the virtual corner line at (x, y) cuts out exactly an area of size a (Figure 3.12). We can show that if there does not exist any single virtual corner line for a having sufficiently small length, then there exist virtual corner lines for a at two points (x_1, y_1) and (x_2, y_2) such that the former has short (i.e. at

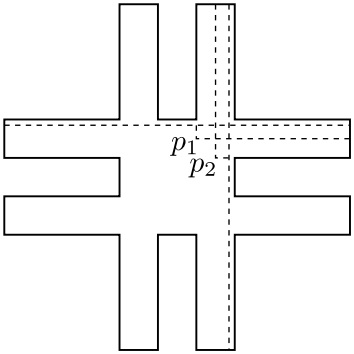


Figure 3.11: *A polygon in which every virtual corner line for a is too long. At p_1 the vertical length switches from short to long and at p_2 the horizontal length switches from long to short.*

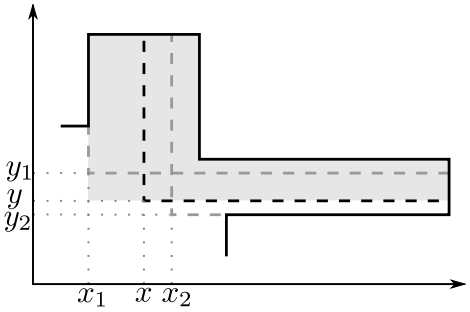


Figure 3.12: *The interval $[x_1, x_2]$ in a polygon and a virtual corner line (dashed black) for a whose horizontal and vertical lengths are both large.*

most $c\sqrt{a}$) vertical length, the latter has short horizontal length, and for all virtual corner lines in between both lengths are large.

Lemma 3.17. *Let $\mathscr{P} \subset \mathbb{R}^2$ be an open set of points in the plane of size n, $a \in [0, n]$, and c be a constant. Suppose there is no virtual corner line for a with a length of at most $2c\sqrt{a}$. Then there is an interval $[x_1, x_2] \subseteq I_a$ such that*

- $l_v(x_1) \leq c\sqrt{a}$,
- $l_v(x) > c\sqrt{a}$ *for all* $x \in]x_1, x_2]$,
- $l_h(x_2) \leq c\sqrt{a}$, *and*
- $l_h(x) > c\sqrt{a}$ *for all* $x \in [x_1, x_2[$.

Proof. Let

$$x_2 = \inf\left\{x \in I_a \mid l_h(x) \leq c\sqrt{a}\right\} \text{ and}$$
$$x_1 = \sup\left\{x \in I_a \mid l_v(x) \leq c\sqrt{a} \ \wedge \ x \leq x_2\right\}.$$

We need to show that if the premise holds, i.e. if there is no $x \in I_a$ such that $l(x) \leq 2c\sqrt{a}$, then the interval $[x_1, x_2]$ fulfils the above listed properties. It is easy to see that $l_v(x') = 0$ and $l_h(x'') = 0$, where $I_a = [x', x'']$, and from the premise it then follows that $l_h(x') > 2c\sqrt{a}$ and $l_v(x'') > 2c\sqrt{a}$. Hence the points x_1 and x_2 must exist in I_a since the vertical length must switch from short to long and the horizontal length from long to short when traversing the interval. Assume that $l_v(x_1) > c\sqrt{a}$. Since $\mathscr{P}$ is an open set of points there must then be some $z > 0$ such that $l_v(x) > c\sqrt{a}$ for all $x \in [x_1 - z, x_1]$. However this contradicts the definition of x_1 and we can hence conclude that $l_v(x_1) \leq c\sqrt{a}$. A similar argument can be given for $l_h(x_2)$ and thus also $l_h(x_2) \leq c\sqrt{a}$.

The premise states that $l_h(x) + l_v(x) > 2c\sqrt{a}$ for all $x \in I_a$. Thus by the pigeon-hole principle we can conclude that $l_h(x) > c\sqrt{a}$ or $l_v(x) > c\sqrt{a}$ for any such x. By the definition of x_1 and x_2 it therefore holds that $x_1 < x_2$ and for all points $x \in]x_1, x_2[$ it holds that $l_v(x) > c\sqrt{a}$ and $l_h(x) > c\sqrt{a}$. Hence the properties on the horizontal and vertical lengths listed above are true. □

Further we can show that the interval $[x_1, x_2]$ of the above lemma is also short.

Lemma 3.18. *Let $\mathscr{P}$ be a polygon of size n, $a \in [0, n]$, and $c \geq 2$ be a constant. If (x_1, y_1) and (x_2, y_2), where $x_1 < x_2$, are the corners of two virtual corner lines for a in $\mathscr{P}$ such that the interval $[x_1, x_2]$ has the properties listed in Lemma 3.17, then*

$$x_2 - x_1 < \frac{2\sqrt{a}}{c} \text{ and } y_1 - y_2 < \frac{2\sqrt{a}}{c}.$$

Proof. Fix some $x \in]x_1, x_2]$ and let (x, y) be the corner of the virtual corner line $\Lambda(x)$ for a (see Figure 3.12). Let $\mathscr{Q}_y$ be the area cut out by the virtual corner line (for some $a + b$ where $b > 0$) with corner (x_1, y), i.e. both sets $\mathscr{P}(x_1)$ and $\mathscr{P}(x)$ are included in $\mathscr{Q}_y$. We can derive an upper bound on the size of $\mathscr{Q}_y \setminus \mathscr{P}(x)$ by observing that this area can be split into two parts. Of these, one is contained in the area $\mathscr{P}(x_1)$ while the other is contained in the rectangle below this area. Hence we can conclude that the size of $\mathscr{Q}_y \setminus \mathscr{P}(x)$ is at most $a + h(y) \cdot w(x)$, where $h(y) = y_1 - y$ and $w(x) = x - x_1$ are the height and width of the rectangle, respectively.

We can derive a lower bound on the size of $\mathscr{Q}_y \setminus \mathscr{P}(x)$ by integrating along the vertical lengths of the virtual corner lines between x_1 and x, yielding

$$\lim_{z \to x_1} \int_z^x l_v(t)\, dt > w(x) \cdot c\sqrt{a}.$$

The area $\mathscr{Q}_y \setminus \mathscr{P}(x_1)$ is split into the part that is entirely contained in $\mathscr{P}(x)$ and the part that is contained in the rectangle to the left of that area. Note that the latter rectangle is the same as for the area $\mathscr{Q}_y \setminus \mathscr{P}(x)$. Since $\Lambda(x_1)$ and $\Lambda(x)$ both cut out an area of size a, we can conclude that the sizes of the areas $\mathscr{P}(x) \setminus \mathscr{P}(x_1)$ and $\mathscr{P}(x_1) \setminus \mathscr{P}(x)$ are equal. Therefore also the areas $\mathscr{Q}_y \setminus \mathscr{P}(x)$ and $\mathscr{Q}_y \setminus \mathscr{P}(x_1)$ have the same size. Hence we can derive a second lower bound of $h(x) \cdot c\sqrt{a}$ on the size of $\mathscr{Q}_y \setminus \mathscr{P}(x)$ by integrating along the horizontal lengths of the virtual corner lines in a similar way as before. By defining $l(x) = \max\{w(x), h(x)\}$ we can thus conclude that the size of $\mathscr{Q}_y \setminus \mathscr{P}(x)$ is greater than $l(x) \cdot c\sqrt{a}$. By using $l(x)$ as an estimate

of $h(x)$ and $w(x)$ in the upper bound derived above, we get the following inequality:

$$a + l(x)^2 > l(x) \cdot c\sqrt{a}$$

This inequality is an invariant that is true for any $x \in]x_1, x_2]$. Using standard methods we can derive that the two terms in the invariant are equal if

$$l(x) = \frac{\sqrt{a}}{2}\left(c \pm \sqrt{c^2 - 4}\right).$$

Since $c \geq 2$ this means that the invariant amounts to one of the following terms:

$$l(x) < \frac{\sqrt{a}}{2}\left(c - \sqrt{c^2 - 4}\right) \text{ or} \tag{3.1}$$

$$l(x) > \frac{\sqrt{a}}{2}\left(c + \sqrt{c^2 - 4}\right). \tag{3.2}$$

Note that this means that there is an interval between these two bounds from which $l(x)$ cannot take a value. However, since the vertical and horizontal lengths of $\Lambda(x)$ are always greater than zero for $x \in]x_1, x_2[$, both $w(x)$ and $h(x)$ are continuous functions in the interval $]x_1, x_2]$. Therefore also $l(x)$ is a continuous function. Since $l(x)$ can be arbitrarily close to zero this means that Inequality (3.2) can never be fulfilled.

It is easily verifiable that the right-hand side of Inequality (3.1) can be upper-bounded by $\frac{2}{c}\sqrt{a}$ since $c \geq 2$. The proof is concluded by noticing that an upper bound on $l(x_2)$ is also one for $w(x_2)$ and $h(x_2)$. □

Analogous to a virtual corner line we can define a *virtual staircase line* by considering any staircase line of infinite length in the plane and taking the parts of the line that lie inside some specific polygon.

Definition 3.19 (virtual staircase line)**.** Let μ be a staircase line in the plane $\mathbb{R}^2$ of orientation down. For any finite area $\mathscr{P} \subset \mathbb{R}^2$ the set Λ of staircase, corner, and straight lines in $\mathscr{P}$ for which $\lambda \in \Lambda$ if and only if $\lambda \subset \mu \cap \mathscr{P}$ is called a *virtual*

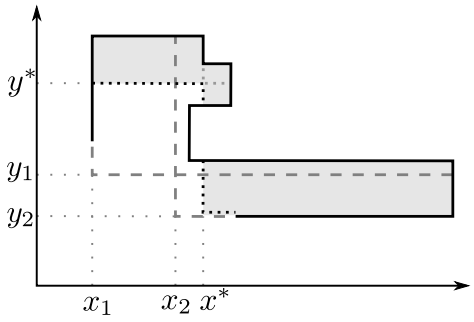


Figure 3.13: *A virtual staircase line (dotted black) cutting out the area of size a shaded in grey. It is constructed using the two virtual corner lines at (x^*, y_2) and (x_1, y^*).*

staircase line. The *length* of Λ is the sum of the lengths of the included straight, corner, and staircase lines. If Λ cuts out an area of size a on its upper right side we say that it is a virtual corner line *for* a.

Notice that a virtual corner line is also a virtual staircase line. Using the above results we find a virtual staircase line which cuts out exactly the required area a and has a short total length (Figure 3.13). The corresponding staircase line goes along the vertical section of the first virtual corner line, to some y^* and then turns to the right and goes to some x^*, turns again and then finally follows the horizontal part of the second virtual corner line.

Lemma 3.20. *Given a polygon $\mathscr{P}$ of size n, for any $a \in [0, n]$ there is a virtual staircase line Λ for a that has a length of at most $7\sqrt{a}$.*

Proof. We attempt to cut out an area of size a from $\mathscr{P}$ using a virtual corner line. Due to Lemmas 3.17 and 3.18 we can either find one with length at most $4\sqrt{a}$ (throughout this proof we set $c = 2$) or there is an interval $[x_1, x_2]$ with the properties listed in the lemmas. In the former case let Λ contain this set of lines. In the latter case we can find the desired set of curves as follows

(Figure 3.13). We will use the same notation as in the proof of Lemma 3.18.

For any $x \geq x_2$ let $\Lambda'(x)$ be the virtual corner line with corner (x, y_2) and let $l'_v(x)$ be its vertical length. We attempt to find $x^* = \min\{x \geq x_2 \mid l'_v(x) \leq \sqrt{a}\}$ (which is well defined since $\mathscr{P}$ is an open set). Notice that the vertical lines that are part of $\Lambda'(x)$ are contained in $\mathscr{P}(x_2)$, which has size a. Hence by the definition of x^* we can conclude that

$$a \geq \lim_{z \to x^*} \int_{x_2}^{z} l'_v(x)\, dx > (x^* - x_2)\sqrt{a},$$

which means that $x^* - x_2 < \sqrt{a}$.

Let $\mathscr{P}'(x^*) \subset \mathscr{P}(x_2)$ be the area that is cut out by $\Lambda'(x^*)$. We define y^* to be the coordinate where the virtual corner line with corner (x_1, y^*) cuts out an area $\mathscr{Q}$ such that $\mathscr{Q} \cup \mathscr{P}'(x^*)$ has size a. Observe that $y^* \geq y_1$ since $\mathscr{P}(x_1)$ has size a and hence $\mathscr{Q} \subseteq \mathscr{P}(x_1)$. The desired set of curves Λ contains all curves λ that are segment curves and

$$\begin{aligned}
\lambda \subseteq &\{(x, y_2) \in \mathscr{P} \mid x \geq x^*\} \cup && (3.3)\\
&\{(x^*, y) \in \mathscr{P} \mid y \in [y_2, y^*]\} \cup && (3.4)\\
&\{(x, y^*) \in \mathscr{P} \mid x \in [x_1, x^*]\} \cup && (3.5)\\
&\{(x_1, y) \in \mathscr{P} \mid y \geq y^*\}. && (3.6)
\end{aligned}$$

The points in the first set (in Row (3.3)) are contained in the horizontal parts of $\Lambda(x_2)$ and the points in the last set (in Row (3.6)) are contained in the vertical parts of $\Lambda(x_1)$, which each have a length of at most $2\sqrt{a}$ by Lemma 3.17. The points in the second set (in Row (3.4)) are contained in the vertical parts of $\Lambda'(x^*)$ which by definition has a length of at most $\sqrt{a}$. The length of the parts from the third set (in Row (3.5)) are at most the distance between x_1 and x^*. By Lemma 3.18 the distance between x_1 and x_2 is at most $\sqrt{a}$. By the observations above the distance from x_2 to x^* is also at most $\sqrt{a}$. In total this gives a length of at most $7\sqrt{a}$ for the curves in Λ. □

After replacing the rectangular line we will see that we are left with a set L of straight and corner lines cutting out an area

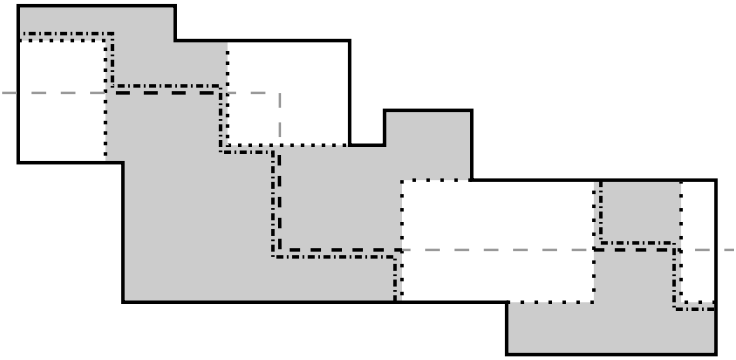


Figure 3.14: *A virtual staircase lined (dashed) that is converted to a set of staircase, corner, and straight lines (dashed and dotted). For this, parts of the corner lines (dotted) from the original cut are used. These are all concave w.r.t. the same part of the cut shaded in grey.*

$m-a$ or $m+a$. We now know that there exists a virtual staircase line Λ that can be used to cut out the remaining area of size a from the $\mathscr{A}$- or $\mathscr{B}$-part. Notice that the underlying staircase line (of infinite length in the plane) may be intersecting with other curves in the cut (Figure 3.14). So the parts of the line included in Λ may not have endpoints on the boundary of the polygon. Thus, we need to convert Λ into a set M of staircase, corner, and straight lines, none of which ends at any other curve in L (however, the curves may partially overlap). This is done by adding those parts of curves in L to the curves in Λ that are monotonic extensions of the latter in x- and in y-direction. This is possible since the corner lines in L are all concave w.r.t. the same cut out part, as pointed in the previous section. Thus the set M may contain several staircase lines, but its total length is at most that of L.

Lemma 3.21. *For any polygon $\mathscr{P}$, let L be a non-crossing corner m-cut with cut size C, such that all corner lines in L are concave with respect to $\mathscr{A}(L)$. For any $a \in [0, m]$ there is a set of segment curves M in $\mathscr{P}$ that cuts out an area of size a from $\mathscr{A}(L)$ and has the following properties. The set $M \cup L$*

is non-crossing, M contains only straight, corner, and staircase lines, and the cut size of M is at most $7\sqrt{a} + C$. Furthermore any staircase line in M is oriented down and its surplus, w.r.t. the $(m-a)$-cut $M \cup L$, lies on its lower left side.

Proof. By Lemma 3.20 we can find a virtual staircase line Λ in $\mathscr{A}(L)$ that cuts out an area of size a and has a length of at most $7\sqrt{a}$. The boundary points of a line $\lambda \in \Lambda$ with respect to $\mathscr{A}(L)$ are either boundary points with respect to $\mathscr{P}$ or they are points on curves in L. If there is a $\lambda' \in L$ and a point $(x, y) \in \lambda'$ such that (x, y) is a boundary point of λ w.r.t. $\mathscr{A}(L)$, the assumption that all corner lines in L are concave w.r.t. $\mathscr{A}(L)$ lets us conclude that λ' lies on the opposite side of (x, y) than λ does. More formally, there is a relation $\lesseqgtr \in \{\leq, \geq\}$ such that for all $(x', y') \in \lambda'$ and all $(x'', y'') \in \lambda$ it either holds that $x' \lesseqgtr x$ while $x \lesseqgtr x''$ or that $y' \lesseqgtr y$ while $y \lesseqgtr y''$. Let in the former case $\mu_{(x,y)} = \{(x', y') \in \lambda' \mid y \lesseqgtr y'\}$ and in the latter case $\mu_{(x,y)} = \{(x', y') \in \lambda' \mid x \lesseqgtr x'\}$. That is, if λ lies to the right or to the left of (x, y) the set $\mu_{(x,y)}$ contains the parts of λ' above or below (x, y), respectively, and if λ lies above or below (x, y) the set $\mu_{(x,y)}$ contains the parts of λ' to the right or to the left of (x, y), respectively.

To construct the desired a-cut M in $\mathscr{P}$ we initially set $M = \Lambda$. If γ denotes the boundary of the area of size a that Λ cuts out, we add to M any curve in L that is contained in γ. Let P denote the set of boundary points of the curves in M w.r.t. $\mathscr{A}(L)$ which are contained in some curve from L. Since Λ, and hence initially also M, is a virtual staircase line in $\mathscr{A}(L)$, for any straight line $\lambda' \in L$ there can be at most one curve in M that has a boundary point on λ'. For any corner line $\lambda' \in L$ there can be at most two lines $\lambda_1, \lambda_2 \in M$ that have boundary points p and q on λ'. Of these points one must be on the vertical and one on the horizontal part of λ'. Hence the sets μ_p and μ_q intersect. In this case we replace the lines λ_1 and λ_2 by the line $\lambda_1 \cup \lambda_2 \cup (\mu_p \cap \mu_q)$ in M. At the same time we remove the points p and q from the set P. We repeat this process until no pair of points in P remain that both are part of some single line $\lambda' \in L$. For any remaining point in P we now know that if it is contained in some curve $\lambda' \in L$ then it is the only one. For any such point p we replace

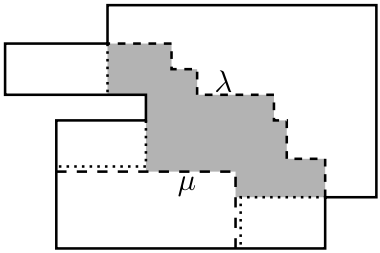


Figure 3.15: *A staircase line λ with its surplus shaded in grey. The curves on the boundary of the surplus can be replaced by a set of straight and corner lines (dotted). The corner line μ is also removed.*

the line $\lambda \in M$, for which p is a boundary point, with the line $\lambda \cup \mu_p$ in M and remove p from P. This is repeated until no points remain in the set P.

Since the curves in L are straight lines or corner lines that are concave w.r.t. $\mathscr{A}(L)$, the added parts of the curves in L connect the curves in the original set Λ with the boundary of the polygon $\mathscr{P}$ in such a way that in the end M contains only straight, corner, or staircase lines. Notice that the latter lines are all oriented down by the fact that Λ is oriented down and by the definition of the sets μ_p for $p \in P$. Furthermore the area $\mathscr{B}(M \cup L)$ of the $(m-a)$-cut $M \cup L$ contains the parts of $\mathscr{A}(L)$ that were cut out by the virtual staircase line Λ on its upper right side. Hence the surplus, defined w.r.t. the $(m-a)$-cut $M \cup L$, of a staircase line in M must lie on its lower left side. Finally each added part from a curve in L was only added once to a staircase line while constructing M. This means that the total length of the curves in M is at most $7\sqrt{a} + C$. □

The next step is to convert the staircase lines from the set $M \cup L$ so that at most one of them remains but the cut size does not increase. Similar to the techniques seen before, we will use the curves contained in the boundary of the surplus or deficit of a staircase line for the transformation. Unfortunately some of the previous arguments can not be used here since $M \cup L$ is

not an optimal cut. Instead we need some observations on the nature of the boundary of the deficit and surplus of a staircase line $\lambda \in M$: it turns out that any staircase line λ' different from λ at the boundary of the deficit or surplus of λ overlaps with exactly one corner line $\mu \in L$ (Figure 3.15). This corner line μ together with the staircase line λ' can be used to construct a pair of corner lines. These can be replaced with μ and λ' so that the same region is cut out by the new set of curves. The cut size decreases during this process.

Lemma 3.22. *For a polygon $\mathscr{P}$, let L be a set of non-crossing straight and corner lines and λ be a staircase line that does not cross any curve in L. Let Λ denote the set of segment curves in $\mathscr{P}$ that are contained in the boundary of λ's surplus (deficit), apart from λ itself, where the surplus (deficit) is defined w.r.t. the cut $L \cup \{\lambda\}$. If the set $L \cup \Lambda$ cuts out an area of size m then there exists an m-cut that has a cut size at most that of $L \cup \Lambda$ and contains only straight and corner lines.*

Proof. We will prove the statement for the case when the curves in Λ are contained in the boundary of λ's surplus. The other case is analogous. If Λ only contains straight and corner lines the lemma obviously holds. By the definition of the surplus the only problem that can arise is when Λ contains a staircase line λ'. Assume w.l.o.g. that λ is oriented down, which means that also λ' is. Assume furthermore that the surplus of λ lies on the lower left side of λ. If we partition λ' into a succession of bar lines that are alternating horizontally and vertically oriented, λ' consists of at least three successive bar lines since it is a staircase line. Hence there must be a horizontal bar line σ_h that, to its right, is followed by a vertical bar line σ_v. Let (x, y) denote the point at which these two bar lines meet. This means that σ_h lies to the left of (x, y) and σ_v below (x, y). In this sense (x, y) is a *concave corner* of the boundary of λ's surplus. We want to argue that there can be at most one such point and it is the corner of a corner line from L. These facts can then be used to convert λ' into a set of appropriate corner lines.

Since (x, y) is part of the boundary of λ's surplus, we know that for any $z > 0$ the point $(x-z, y-z)$ to the lower left of (x, y)

is not part of the surplus. Since the point (x, y) is a concave corner of the surplus, for any sufficiently small z and $z_x, z_y \geq 0$ there must be two points $(x - z, y + z_y)$ and $(x + z_x, y - z)$, i.e. to the top left and the lower right of (x, y), that are part of the surplus. (It holds that $z_x = 0$ or $z_y = 0$ if the respective point lies on λ. This may happen since λ is part of the surplus.) If z is small enough then there is no point $(x - z, y')$ or $(x', y - z)$, for any $y' \in [y - z, y + z_y]$ and $x' \in [x - z, x + z_x]$, that lies on the boundary of the polygon $\mathscr{P}$. Hence the only reason why $(x - z, y - z)$ is not part of the surplus can be that the point lies in $\mathscr{B}(L \cup \{\lambda\})$ and not in $\mathscr{A}(L \cup \{\lambda\})$. Letting z tend to zero it follows that (x, y) must be part of some curve $\mu \in L$ that cuts out the area to which the point $(x - z, y - z)$ belongs. Obviously μ is a corner line with corner (x, y) which includes the horizontal and vertical bar lines σ_h and σ_v.

Suppose there are more than one concave corner of λ'. Then there must be at least two of these that are adjacent in the sense that the vertical bar line σ_v^p of one of the corners p shares a point r with the horizontal bar line σ_h^q of the other corner q. By the arguments given above there must be two corner lines μ^p and μ^q in L such that $\sigma_v^p \subset \mu^p$ and $\sigma_h^q \subset \mu^q$. But since r is not part of the boundary of $\mathscr{P}$ this means that μ^p and μ^q cross at this point, which is a contradiction. Hence there can only be one concave corner of λ'. In particular this means that σ_h is the only horizontal bar line of λ' that has an adjacent vertical bar line to its right while σ_v is the only vertical bar line that has an adjacent horizontal bar line to its left.

Consider the case when there is a horizontal bar line σ_h' to the right of the vertical bar line σ_v. As noted above, σ_h' must have a boundary point. Removing σ_v from the corner line μ that overlaps with λ' leaves the horizontal bar line of μ and some vertical bar line σ_v' that is the lower extension of σ_v in μ. Obviously σ_v' has a boundary point. Hence by removing σ_v from both μ and λ' we can construct a corner line $\sigma_h' \cup \sigma_v'$. Similarly the horizontal bar line σ_h can be removed from λ' and μ, leaving a corner line if there is a vertical bar line above σ_h in λ'. If λ' and μ share a boundary point then removing σ_v or σ_h as described above obviously leaves nothing to be taken care of.

Hence any staircase line in the set Λ can, together with some corner line from L, be replaced with one or two corner lines. In a cut that includes the curves from Λ and L this transformation will not change the size of the cut out area and will decrease the cut size. □

The above observations can now be used to convert the staircase lines constructed in Lemma 3.21 in such a way that only one staircase line remains, as the next lemma shows.

Lemma 3.23. *In a polygon $\mathscr{P}$ let L be a non-crossing corner m-cut with cut size C such that all corner lines in L are concave with respect to $\mathscr{A}(L)$. Then for any $a \in [0, m]$ there exists a $(m-a)$-cut M in $\mathscr{P}$ with cut size at most $7\sqrt{a} + 2C$ that contains only straight and corner lines except at most one which is a staircase line.*

Proof. By Lemma 3.21 we can find a set of curves M' such that $L \cup M'$ is a $(m-a)$-cut that fulfils all properties of the statement except for the fact that $L \cup M'$ may contain more than one staircase line. Due to the additional properties that any staircase line is oriented down and its surplus lies on the lower left side, we can conclude that the boundaries of the surplus and the deficit can not contain any other staircase line. Hence we may use Lemma 3.22 to convert a staircase line into a set of straight and corner lines as follows.

We initially set $M = L \cup M'$. Let λ_1 and λ_2 be two distinct staircase lines from M', and let b_1 be the size of the surplus of λ_1 and b_2 be the size of the deficit of λ_2. Without loss of generality we can assume that $b_1 \leq b_2$. Similar to the proof of Lemma 3.14 we can find two sets of curves L_1 and L_2 that cut out an area of size b_1 from the surplus of λ_1 and the deficit of λ_2, respectively, such that removing λ_i and adding L_i, for both $i \in \{1, 2\}$, in M' again yields a $(m-a)$-cut M. The set L_2 can be chosen to consist of a single staircase line if $b_2 > b_1$ and it contains only curves that are part of the boundary of λ_2's deficit if $b_2 = b_1$. The set L_1 always contains curves that are part of the boundary of λ_1's surplus. The new $(m-a)$-cut in which λ_1 and λ_2 were replaced has a cut size that is at most the cut size

of the old m-cut since distances are measured in the l_1-norm (it is decreasing if there are more than one curve in L_1 or L_2 since then parts of the boundary of $\mathscr{P}$ act as a short cut for the curves).

Using Lemma 3.22 the staircase lines in L_1 can all be converted to corner and straight lines. If there are more than one staircase line in L_2, i.e. L_2 is part of the boundary of λ_2's deficit, using the same lemma all of them can be converted to straight and corner lines. Otherwise L_2 consists of only one staircase line. Hence repeating this procedure with any remaining pair of staircase lines in M will eventually yield a $(m-a)$-cut in which there is at most one staircase line left. Since the cut size is non-increasing during each transformation step, the cut size of the final set M is at most $7\sqrt{a}+2C$, which concludes the proof. □

Using the above techniques we can find an m-cut containing at most one staircase line for any optimal m-cut containing a rectangular line, such that the cut size of the former m-cut is at most a constant times the cut size of the latter. The following theorem summarises these results.

Theorem 3.24. *For any polygon $\mathscr{P}$ with an optimal m-cut L of $\mathscr{P}$ containing a rectangular line, there exists a non-crossing m-cut M which contains only corner and straight lines except at most one which is a staircase line and M has a cut size of at most $9C$, where C is the cut size of L.*

Proof. Let $\rho \in L$ be a rectangular line in L which w.l.o.g. is concave w.r.t. $\mathscr{A}(L)$. Let $\mathscr{R}$ be the defining rectangle of ρ and let $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ be the boundary points of ρ. We consider $\mathscr{R}$ to be a closed set, i.e. $\mathscr{R}$ contains its boundary. Assume w.l.o.g. that ρ is oriented in a way such that p_1 is part of the left boundary of $\mathscr{R}$ while p_2 is part of the lower boundary of $\mathscr{R}$. This in particular means that $x_1 \leq x_2$ and $y_1 \geq y_2$. Let β be the boundary of the polygon $\mathscr{P}$ and

$$x_{\max} = \max\{x \in \mathbb{R} \mid (x, y) \in \beta \cap \mathscr{R}\} \text{ and}$$
$$y_{\max} = \max\{y \in \mathbb{R} \mid (x, y) \in \beta \cap \mathscr{R}\}$$

be the extreme points of the boundary β in $\mathscr{R}$. Notice that $x_{\max} \geq x_2$ and $y_{\max} \geq y_1$ since p_1 and p_2 are boundary points and hence belong to β and (the boundary of) $\mathscr{R}$. We define the set of curves Ξ (Figure 3.9) such that $\lambda \in \Xi$ if and only if λ is a segment curve and

$$\begin{aligned}\lambda \subseteq \{(x_{\max}, y) \in \mathscr{P} \mid y \in [y_2, y_{\max}]\} \cup \\ \{(x_1, y) \in \mathscr{P} \mid y \in [y_1, y_{\max}]\} \cup \\ \{(x, y_{\max}) \in \mathscr{P} \mid x \in [x_1, x_{\max}]\} \cup \\ \{(x, y_2) \in \mathscr{P} \mid x \in [x_2, x_{\max}]\}.\end{aligned}$$

The set Ξ can be seen as a virtual rectangular line.

Let $\mathscr{R}'$ be the "defining rectangle" of Ξ, i.e. the rectangle that is defined by the two opposing corners (x_1, y_2) and $(x_{\max}, y_{\max})$. There are three corners of $\mathscr{R}'$ that some curve in Ξ might include, namely $(x_1, y_{\max})$, $(x_{\max}, y_2)$, and $(x_{\max}, y_{\max})$. If $(x_1, y_{\max})$ is included in some curve $\lambda \in \Xi$ then this point cannot be a boundary point. Hence it must be the case that $y_{\max} > y_1$ and thus, by the definition of $y_{\max}$, there is some part of β that intersects with the upper boundary of $\mathscr{R}'$. But then there can be no single curve in Ξ that contains both $(x_1, y_{\max})$ and $(x_{\max}, y_{\max})$ since these are the endpoints of the upper boundary of $\mathscr{R}'$. A similar argument holds for $(x_{\max}, y_2)$ and $(x_{\max}, y_{\max})$. Therefore no curve in Ξ contains more than one corner, i.e. Ξ includes only straight and corner lines.

Let $\mathscr{D} \subseteq \mathscr{R}$ be the area that is cut out between ρ and Ξ in the a-cut $\Xi \cup \{\rho\}$, where a is the size of $\mathscr{D}$. (Remember that this means that $\mathscr{D}$ is an open set.) Due to Lemma 3.13, apart from ρ the set L contains only straight and corner lines. Hence, since no curve crosses ρ and because by the construction of Ξ the set $\mathscr{D}$ does not intersect with the boundary β, no curve from L crosses a curve in Ξ. Thus we can replace ρ with Ξ in L and yield a non-crossing $(m + a)$-cut M'. By the construction of Ξ the corner lines in Ξ are concave w.r.t. $\mathscr{A}(M') = \mathscr{A}(M) \cup \mathscr{D}$, and by Lemma 3.13 the corner lines in L are concave w.r.t. $\mathscr{A}(M)$. Hence all corner lines in M' are concave w.r.t. $\mathscr{A}(M')$. By Lemma 3.23 we can thus find an m-cut M that has a cut size of at most $7\sqrt{a} + 2C'$, where C' is the cut size of M', such that

M contains only straight and corner lines except for at most one which is a staircase line.

By the construction of Ξ the total length of the curves in Ξ is at most the length of ρ and hence the cut size of M' is at most the cut size of L, i.e. $C \geq C'$. At the same time, since the size a of $\mathscr{D} \subseteq \mathscr{R}$ is smaller than the size of $\mathscr{R}$, and since the length l_ρ of ρ is greater than the height of $\mathscr{R}$ plus the width of $\mathscr{R}$, it follows that $C \geq l_\rho \geq \sqrt{a}$. Hence we can upper bound the cut size of M by $7\sqrt{a} + 2C' \leq 9C$, which proves the claim. □

3.2.4 Removing Staircase Lines

We now turn to the task of converting a (not necessarily optimal) cut L containing only straight and corner lines except one which is a staircase line, into a cut containing only straight and corner lines. Similar to the case of rectangular lines we will replace the staircase line with a set of appropriate corner and straight lines along the boundary of the deficit (or surplus). It is easy to see that if the deficit (or surplus) area of the staircase line λ has size a, then $\sqrt{a} < C$, where C is the cut size of L. Thus, if we can cut out the excess area a using straight and corner lines of total length in $\mathcal{O}(\sqrt{a})$, then our cut size will still be close to optimal. Given any simple polygon $\mathscr{P}$ of area n, $a \in [0, n]$, and $\varepsilon \in]0, 1]$ we can find a set of at most three virtual corner lines that cut out an area whose size is in the interval $[(1-\varepsilon)a, (1+\varepsilon)a]$ with a cut size that is a constant (depending on ε) times $\sqrt{a}$. Furthermore the corners of these virtual corner lines all have either the same x-coordinate or the same y-coordinate. They can be found using the short interval $[x_1, x_2]$ that was identified before (Figure 3.12). We use the virtual corner line with corner (x_1, y_2) which has short length but cuts out an area that is too large. To correct for the area we additionally find two virtual corner lines (of short length) with corners at either points (x', y_2) and (x'', y_2), for some $x', x'' \geq x_2$, or points (x_1, y') and (x_1, y''), for some $y', y'' \geq y_1$.

Lemma 3.25. *For any polygon $\mathscr{P}$ of total area n, any $a \in [0, n]$, and any $\varepsilon \in]0, 1]$ there is a set L of straight and corner lines with the following properties. The lines in L cut out an area*

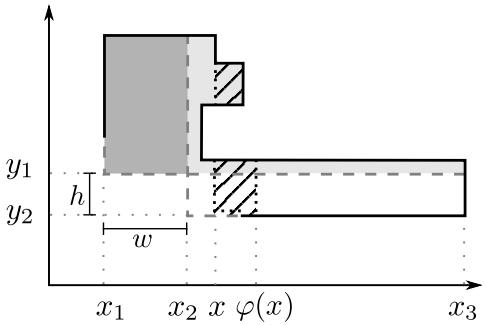


Figure 3.16: *The interval $[x_1, x_2]$ of width w together with the virtual corner lines $\Lambda(x_1)$ and $\Lambda(x_2)$ (grey dashed lines). The dark grey area is $\mathscr{P}(x_1) \setminus \mathscr{P}(x_2)$ of size b', while the light grey area is $\mathscr{P}(x_1) \cap \mathscr{P}(x_2)$ of size b. The right-most point of $\mathscr{P}(x_2)$ is x_3, and x and $\varphi(x)$ define points at which the virtual corner lines $\Lambda'(x)$ and $\Lambda'(\varphi(x))$ (black dotted lines) enclose an area of size d' between them (striped pattern).*

which has a size in the interval $[(1-\varepsilon)a, (1+\varepsilon)a]$, and the cut size of L is at most $(6\sqrt{7/\varepsilon}+2)\cdot\sqrt{a}$. Furthermore L is the union of at most three virtual corner lines with corners that either have the same x- or y-coordinate.

Proof. If ε and a are chosen such that $(1+\varepsilon)a \geq n$ the lemma trivially holds since L can be empty. Hence assume that $(1+\varepsilon)a < n$ throughout this proof. Let $c = \sqrt{7/\varepsilon}$. If there is a virtual corner line for a that has a length of at most $2c\sqrt{a}$ the lemma obviously holds. Since $\sqrt{7/\varepsilon} > 2$ for $\varepsilon \in]0, 1]$, if there is no such virtual corner line then by Lemmas 3.17 and 3.18 there is an interval $[x_1, x_2] \subseteq I_a$ with the properties listed therein. We use the same notation as in the proof of Lemma 3.18, but for better readability let $\mathscr{Q} = \mathscr{Q}_{y_2}$, $w = w(x_2)$, and $h = h(y_2)$. The size of the area $\mathscr{Q}$ is $a + d$ for some $d > 0$, i.e. the size of the area $\mathscr{Q} \setminus \mathscr{P}(x_2)$ is d (see Figure 3.16).

Our first goal in this proof is to establish an upper bound on the size b of the area $\mathscr{P}(x_1) \cap \mathscr{P}(x_2)$ depending on d. For this

we establish a lower bound on the size b' of $\mathscr{P}(x_1) \setminus \mathscr{P}(x_2)$ which we can then subtract from a, the size of $\mathscr{P}(x_1)$. One simple bound can be derived by subtracting from d the size of the area not in $\mathscr{P}(x_1) \setminus \mathscr{P}(x_2)$ but in $\mathscr{Q} \setminus \mathscr{P}(x_2)$. Since the size of this area can be upper-bounded by $h \cdot w$, we get $b' \geq d - hw$. We can derive an upper bound for w depending on d by integrating along the vertical lengths of the virtual corner lines for a between x_1 and x_2. By Lemma 3.17 this gives

$$d \geq \lim_{z \to x_1} \int_z^{x_2} l_v(t)\, dt > w \cdot c\sqrt{a}.$$

Hence $w < \frac{d}{c\sqrt{a}}$, and using the upper bound on h given by Lemma 3.18 we can conclude that $b' > d\left(1 - \frac{2}{c^2}\right)$. This directly translates into the upper bound on the size of the area $\mathscr{P}(x_1) \cap \mathscr{P}(x_2)$ which is

$$b = a - b' < a - d\left(1 - \frac{2}{c^2}\right). \tag{3.7}$$

The next step is to find a lower bound on b (which also depends on c) under the assumption that no appropriate set L exists. We will show that for $c = \sqrt{7/\varepsilon}$ the upper and lower bounds contradict each other. Let Δ denote the virtual corner line for $a+d$ with corner (x_1, y_2), i.e. Δ cuts out $\mathscr{Q}$. If $d \leq \varepsilon a$ we can cut out an area which has a size in the interval $[a, (1+\varepsilon)a]$ by letting $L = \Delta$. By Lemmas 3.17 and 3.18 the virtual corner line Δ has a length of at most $2(c\sqrt{a} + 2\sqrt{a}/c) \leq (2\sqrt{7/\varepsilon} + 2)\sqrt{a}$ and hence in this case L satisfies the lemma. Therefore let $d > \varepsilon a$ in the remainder of the proof. We will attempt to find L by either including Δ in L and cutting out an area of size approximately d from $\mathscr{Q}$, or by cutting out an area of size approximately a from $\mathscr{Q}$ directly. The decision on whether to include Δ in L is determined by distinguishing between small and big values for d. In case $d < a$ we include Δ in L and otherwise not.

Since we have the freedom to choose the size of the area that we cut out from the interval $[(1-\varepsilon)a, (1+\varepsilon)a]$, we attempt to cut out the smallest possible area from $\mathscr{Q}$. Hence if d' denotes the size of this area, let $d' = \min\{d, a\} - \varepsilon a$. Notice that d' is well-defined since $d > \varepsilon a$, and that the size of the cut out area is

$a - \varepsilon a$ if Δ is not included in L and it is $a + d - (d - \varepsilon a) = a + \varepsilon a$ otherwise. Hence the size of the cut out area lies in the given interval.

To cut out the area of size d' from $\mathscr{Q}$ we use a pair of virtual corner lines such that for both lines either the horizontal parts overlap with $\Lambda(x_2)$ or the vertical parts overlap with $\Lambda(x_1)$. Notice that such a pair always exists, since $d' < a$ and the lines $\Lambda(x_1)$ and $\Lambda(x_2)$ each cut out an area of size a. Since by Lemma 3.17 both the horizontal length of $\Lambda(x_2)$ and the vertical length of $\Lambda(x_1)$ is short, we only have to guarantee that either the vertical or the horizontal lengths of the two desired virtual corner lines are short, respectively. We will concentrate on the case where we pick the virtual corner lines from those that overlap with $\Lambda(x_2)$, since the other case is analogous. Therefore, if x_3 is defined such that $x_3 - x_2$ is the width of $\mathscr{P}(x_2)$, let $\Lambda'(x)$ denote the virtual corner line with corner (x, y_2) for any $x \in]x_2, x_3]$ and let $l'_v(x)$ be its vertical length.

Let $\varphi(x)$ be a function that, for a given virtual corner line $\Lambda'(x)$, gives the x-coordinate of $\Lambda'(\varphi(x))$, such that $\Lambda'(x)$ and $\Lambda'(\varphi(x))$ enclose an area of size d' between them and $\varphi(x) > x$. This means that $\int_x^{\varphi(x)} l'_v(t)\ dt = d'$ and that the domain of φ is upper-bounded by $\varphi^{-1}(x_3)$ where φ^{-1} is the inverse function of φ (notice that the function φ is bijective). Assume that there is no pair of virtual corner lines $\Lambda'(x)$ and $\Lambda'(\varphi(x))$ for which both vertical lengths are shorter than $c\sqrt{a}$. From this assumption it follows that for all $x \in [x_2, \varphi^{-1}(x_3)]$ it holds that $l'_v(x) > c\sqrt{a}$ or $l'_v(\varphi(x)) > c\sqrt{a}$.

Let for any interval $I \subseteq [x_2, x_3]$ the function f be equal to the size of the area $\{(x', y') \in \mathscr{Q} \mid x' \in I\}$ in the vertical stripes in $\mathscr{Q}$ defined by I, i.e.

$$f(I) = \int_I l'_v(t)\ dt.$$

Let $J = \{x \in [x_2, \varphi^{-1}(x_3)] \mid l'_v(x) > c\sqrt{a}\}$ be the subset of the domain of φ for which the vertical lengths of the virtual corner lines $\Lambda'(x)$ are long. Let $\overline{J}$ be the set for which the vertical lengths are short, i.e. $\overline{J} = [x_2, \varphi^{-1}(x_3)] \setminus J$. Also let $K = \{x \in [\varphi(x_2), x_3] \mid l'_v(x) > c\sqrt{a}\}$ and $\overline{K} = [\varphi(x_2), x_3] \setminus K$ be

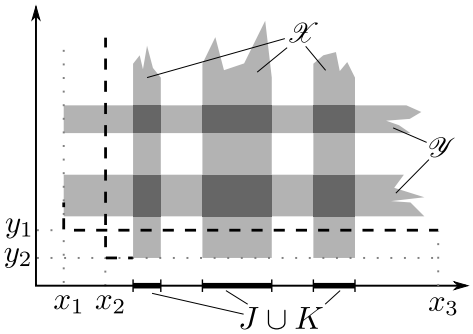


Figure 3.17: *The interval $[x_1, x_2]$ together with the virtual corner lines $\Lambda(x_1)$ and $\Lambda(x_2)$ (dashed lines). To estimate the size b of $\mathscr{P}(x_1) \cap \mathscr{P}(x_2)$ we determine the size $f(J \cup K)$ of $\mathscr{X}$ and the size of $\mathscr{Y}$.*

the corresponding subsets from the domain of φ^{-1}. To establish the connection between the assumption on the lengths of the vertical lines and the lower bound on b we investigate $f(J \cup K)$ (see Figure 3.17).

Let $[l, r] \subseteq \overline{J}$ be a connected subset of $\overline{J}$. By the definitions of f and φ we get

$$\begin{aligned} f([l, r]) &= f([l, \varphi(l)]) + f([\varphi(l), \varphi(r)]) - f([r, \varphi(r)]) \\ &= f([\varphi(l), \varphi(r)]). \end{aligned}$$

Since $\overline{J}$ is a union of connected subsets and φ is bijective we can conclude that $f(\overline{J}) = f(\varphi(\overline{J}))$, where $\varphi(\overline{J})$ is the image of $\overline{J}$. By the assumption that for all $x \in [x_2, \varphi^{-1}(x_3)]$ the vertical length of $\Lambda'(x)$ or of $\Lambda'(\varphi(x))$ is long, $\varphi(\overline{J})$ must be a subset of K and hence $f(\overline{J}) \leq f(K)$. A similar observation can be made for $\overline{K}$, $\varphi^{-1}(\overline{K})$, and J so that $f(\overline{K}) \leq f(J)$.

By the definition of φ we know that $f([\varphi^{-1}(x_3), x_3]) = d'$ and $f([x_2, \varphi(x_2)]) = d'$, while the total area of $\mathscr{P}(x_2)$ has size a. Hence $f(J \cup \overline{J}) = f(K \cup \overline{K}) = a - d'$. From the bounds above and the fact that J and $\overline{J}$ but also K and $\overline{K}$ are disjoint we can

conclude that

$$f(J) + f(K) \geq f(\overline{K}) + f(\overline{J}) = 2(a - d') - f(K) - f(J).$$

The sets J and K might not be disjoint but from the above inequality we get

$$f(J \cup K) + f(J \cap K) = f(J) + f(K) \geq a - d'.$$

By the pigeon-hole principle and the fact that $(J \cap K) \subseteq (J \cup K)$ we can thus conclude that

$$f(J \cup K) \geq \frac{a - d'}{2}.$$

Let $\mathscr{X} = \{(x, y) \in \mathscr{P}(x_2) \mid x \in J \cup K\}$ and let $f_{\mathscr{X}}$ be the size of $\mathscr{X}$, i.e. $f_{\mathscr{X}} = f(J \cup K)$. We now want to also consider the assumption that there is no pair of virtual corner lines amongst those overlapping with $\Lambda(x_1)$ such that both their horizontal lengths are short while cutting out an area of d' between them. Let $\mathscr{Y} \subseteq \mathscr{P}(x_1)$ denote the area such that $(x, y) \in \mathscr{Y}$ if and only if there is a virtual corner line with corner (x_1, y) which has a horizontal length greater than $c\sqrt{a}$, analogous to the definition of $\mathscr{X}$. Using a similar argumentation as for the set $\mathscr{X}$, we can conclude that the size $f_{\mathscr{Y}}$ of $\mathscr{Y}$ must also be at least $\frac{a-d'}{2}$ if no pair of virtual corner lines exists that cuts out an area of size d' such that both horizontal lengths are short.

To yield a lower bound on b we want to consider those parts of $\mathscr{X}$ and $\mathscr{Y}$ that are contained in $\mathscr{P}(x_1) \cap \mathscr{P}(x_2)$ and determine their size. For this we need to find an appropriate bound on the parts of $\mathscr{X}$ and $\mathscr{Y}$ that are not contained in $\mathscr{P}(x_1) \cap \mathscr{P}(x_2)$, but also a bound on the size of the intersection of $\mathscr{X}$ and $\mathscr{Y}$. Therefore let $w_{\mathscr{X}}$ be the total width of $\mathscr{X}$, i.e. $w_{\mathscr{X}}$ is the total length of the interval $J \cup K$. Since $\mathscr{X}$ is contained in $\mathscr{P}(x_2)$ and the latter has a size of a we can conclude that

$$a \geq \int_{J \cup K} l'_v(t)\ dt > w_{\mathscr{X}} \cdot c\sqrt{a},$$

and hence $w_{\mathscr{X}} < \frac{\sqrt{a}}{c}$. If $h_{\mathscr{Y}}$ denotes the total height of $\mathscr{Y}$, a similar argument yields that also $h_{\mathscr{Y}} < \frac{\sqrt{a}}{c}$.

Those parts of $\mathscr{X}$ that are not contained in $\mathscr{P}(x_1)$ are confined to the area below $\mathscr{P}(x_1)$ in $\mathscr{Q}$ which has height h. Hence the area $\mathscr{X} \setminus \mathscr{P}(x_1)$ has a size of at most $h \cdot w_{\mathscr{X}}$. Similarly the area $\mathscr{Y} \setminus \mathscr{P}(x_2)$ has a size of at most $w \cdot h_{\mathscr{Y}}$, since the area to the left of $\mathscr{P}(x_2)$ in $\mathscr{Q}$ has width w. The size of the intersection of $\mathscr{X}$ and $\mathscr{Y}$ can be at most $w_{\mathscr{X}} \cdot h_{\mathscr{Y}}$. Thus, using the bounds on w and h given in Lemma 3.18 we can conclude that

$$b \geq f_{\mathscr{X}} - h \cdot w_{\mathscr{X}} + f_{\mathscr{Y}} - w \cdot h_{\mathscr{Y}} - w_{\mathscr{X}} \cdot h_{\mathscr{Y}} > a - d' - \frac{5}{c^2}a. \quad (3.8)$$

We make a case distinction on the value of d to compare the lower and upper bounds on b. If $d < a$ then $d' = d - \varepsilon a$ so that setting $c = \sqrt{7/\varepsilon}$ in Bounds (3.7) and (3.8) gives

$$b < a - d\left(1 - \frac{2}{7}\varepsilon\right) < a - d + \frac{2}{7}\varepsilon a \text{ and}$$
$$b > a - (d - \varepsilon a) - \frac{5}{7}\varepsilon a = a - d + \frac{2}{7}\varepsilon a,$$

which is a contradiction. In the case when $d \geq a$ it holds that $d' = (1 - \varepsilon)a$ so that, using the fact that $\varepsilon \in]0, 1]$, Bounds (3.7) and (3.8) give

$$b < a - d\left(1 - \frac{2}{7}\varepsilon\right) \leq a - a\left(1 - \frac{2}{7}\varepsilon\right) = \frac{2}{7}\varepsilon a \text{ and}$$
$$b > a - (1 - \varepsilon)a - \frac{5}{7}\varepsilon a = \frac{2}{7}\varepsilon a,$$

which again is a contradiction.

We can thus conclude that one of our assumptions must be wrong. Therefore there always exists a pair of virtual corner lines which cuts out an area of size d' and has a short total length. These can be found either amongst those overlapping with $\Lambda(x_1)$ or those overlapping with $\Lambda(x_2)$. Hence we can find the set L which is the union of these virtual corner lines and, depending on the value of d, also Δ in case we need it. The cut size of L is at most the length of the two corner lines cutting out the area d' between them, plus the length of Δ. Together these three virtual corner lines have a length of at most $4c\sqrt{a} + 2(c + 2/c)\sqrt{a} < \left(6\sqrt{7/\varepsilon} + 2\right)\sqrt{a}$, which concludes the proof. □

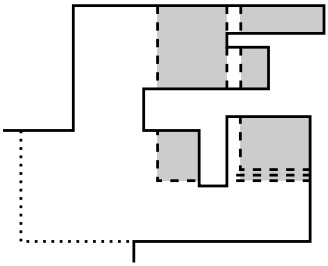


Figure 3.18: *A tail defined by the dotted line. Three (dashed) virtual corner lines cut out the area shaded in grey (the lines overlap on the bottom right).*

Note that in the above proof the existence of $\varepsilon > 0$ guarantees that the intervals J and K have lengths greater than zero: it may happen that $d = a$ so that the size of $\mathscr{Q} \setminus \mathscr{P}(x_1)$ is a. In this case, whether Δ is included in L or not, $\varphi(x_2) = x_3$ if $\varepsilon = 0$. Thus for the proof technique used above we need to allow a deviation from cutting out an exact area size as given by the interval $[(1-\varepsilon)a, (1+\varepsilon)a]$.

To apply the above result, we need to find a region of the polygon of size larger than a that does not contain any curves of the cut, so that we can cut out the excess area without interfering with the other curves. For this we define the concept of a *tail* of a polygon with respect to a cut: for any cut in a polygon $\mathscr{P}$, consider all the connected pieces of the polygon cut out by it. If there a connected piece $\mathscr{T}$ that is defined by a single curve τ then we call $\mathscr{T}$ a tail of the polygon.

Definition 3.26 (tail)**.** For an m-cut L in a polygon $\mathscr{P}$, let $\mathscr{T} \subseteq \mathscr{P} \setminus \bigcup_{\mu \in L} \mu$ be a connected area that is cut out by L. We call $\mathscr{T}$ a *tail* if there exists a single curve $\tau \in L$ that cuts out $\mathscr{T}$. We refer to τ as *the curve of* $\mathscr{T}$. In case L contains a staircase line λ, we call a tail $\mathscr{T} \subseteq \mathscr{A}(L)$ respectively $\mathscr{T} \subseteq \mathscr{B}(L)$ *small* if its area is strictly smaller than λ's deficit respectively surplus.

Notice that there always exists a tail if $\mathscr{P}$ is a simple polygon. Notice also that apart from the curve of a tail $\mathscr{T}$ there might

be other subsets of L that cut out $\mathscr{T}$ if curves in L overlap.

To convert a cut containing a staircase line λ into one containing only straight and corner lines, we can shift λ in either direction, i.e. going into either the $\mathscr{A}$- or the $\mathscr{B}$-part. However all the tails in the polygon may belong to only one part. We need to consider two cases, one of which is when L contains only λ. This means that there are exactly two tails, one on each side of λ. If we assume w.l.o.g. that the size a of λ's deficit is at most that of its surplus, we can replace the staircase line by the set of straight and corner lines on the boundary of its deficit. We then cut out the area $a' \in [(1-\varepsilon)a, (1+\varepsilon)a]$ from the original $\mathscr{A}$-part (containing the surplus) using the at most three virtual corner lines which were shown to exist above (Figure 3.18). The other case is when there is a tail contained in, say, the $\mathscr{A}$-part whose curve μ is not λ. We can safely assume that the size of the tail is larger than the size a of the deficit of λ. If this was not the case then we could remove μ from the cut by using an area exchange with the staircase line λ, without increasing the cut size, as the next lemma shows.

Lemma 3.27. *Let L be an m-cut in $\mathscr{P}$ with cut size C containing one staircase line while all other curves in L are straight and corner lines. There exists an m-cut L' in $\mathscr{P}$ which contains one staircase line and only straight and corner lines otherwise, has cut size of at most C, and the curve of any small tail cut out by L' equals the staircase line.*

Proof. If the curve of every small tail cut out by L is the staircase line $\lambda \in L$ there is nothing to prove. Hence let $\mathscr{T}$ be a small tail cut out by L such that its curve is $\lambda' \neq \lambda$. Assume w.l.o.g. that $\mathscr{T} \subseteq \mathscr{A}(L)$, i.e. $\mathscr{T}$ is strictly smaller than λ's deficit. This means that we can find a staircase line λ'' that cuts out an area that has the same size as $\mathscr{T}$ from λ's deficit, such that removing λ' and replacing λ with λ'' in L yields an m-cut that has a cut size that is less than C by the length of λ'. The new cut has one straight or corner line less than the old one and it contains one staircase line. We repeat this process for any small tail cut out by the new set that does not conform with the desired property. This will eventually terminate in a state in which the resulting m-cut L' fulfils the property. □

As this lemma shows we can replace the staircase line λ by the corner and straight lines on the boundary of its deficit and cut out an area a' from the tail, using the virtual corner lines of short length. It may be that some of the virtual corner lines end at the curve μ of the tail. If this happens we can find a set of straight and corner lines that overlap with parts of the virtual corner lines and μ, with which to replace the latter lines (in the same way as suggested by Figure 3.15). The cut out area is the same while the cut size only grows by a constant factor since there are at most three virtual corner lines. The result of the above described method is summarised in the following theorem.

Theorem 3.28. *Given an m-cut L of a polygon $\mathscr{P}$ with cut size C containing only straight and corner lines except one which is a staircase line, for any desired $\varepsilon \in]0,1]$ there exists a corner m'-cut L', where $m' \in [(1-\varepsilon)m, (1+\varepsilon)m]$, having a cut size of at most $(6\sqrt{7/\varepsilon}+7)\cdot C$.*

Proof. Due to Lemma 3.27 we can assume that any tail cut out by L is not small or its curve is the staircase line $\lambda \in L$. Consider the case when there is a tail $\mathscr{T}$ and its curve is $\lambda' \in L$ such that $\lambda' \neq \lambda$, i.e. $\mathscr{T}$ is not small. In case $\mathscr{T} \subseteq \mathscr{A}(L)$ let a denote the size of λ's deficit and in case $\mathscr{T} \subseteq \mathscr{B}(L)$ let a denote the size of λ's surplus.

The curve λ' is either a straight or a corner line. We assume w.l.o.g. that the horizontal bar line σ'_h of λ' (if any) lies below $\mathscr{T}$ and the vertical bar line σ'_v of λ' (if any) lies to the left of $\mathscr{T}$. That is, for all sufficiently small $z > 0$, $(x_h, y_h) \in \sigma'_h$, and $(x_v, y_v) \in \sigma'_v$ it holds that $(x_h, y_h + z) \in \mathscr{T}$, $(x_h, y_h - z) \notin \mathscr{T}$, $(x_h + z, y_h) \in \mathscr{T}$, and $(x_h - z, y_h) \notin \mathscr{T}$. Notice that this in particular means that if λ' is a corner line then it points up and right if λ' is convex w.r.t. $\mathscr{T}$, and it points down and left if λ' is concave w.r.t. $\mathscr{T}$. According to Lemma 3.25 there is an a'-cut L', for some $a' \in [(1-\varepsilon)a, (1+\varepsilon)a]$, in $\mathscr{T}$ such that L' is the union of at most three virtual corner lines, i.e. L' contains only straight or corner lines where the latter point up and right. Let $\lambda'' \in L'$ be a curve that has a boundary point p with respect to $\mathscr{T}$ such that $p \in \lambda'$. Assume that λ'' is a corner line. If $p \in \sigma'_v$, the corner of λ'' must lie to the right of p since $\lambda'' \subset \mathscr{T}$ and

the assumption made on the location of $\mathscr{T}$ with respect to λ'. However this contradicts the orientation of λ'' since its corner must lie to the left of or below its boundary point. A similar contradiction can be derived if $p \in \sigma'_h$. Hence it must be the case that λ'' is a straight line. If $p \in \sigma'_v$ then let $\sigma \subseteq \sigma'_v$ be the part of σ'_v that lies above p if λ' is a vertical straight line or λ' is a convex corner line w.r.t. $\mathscr{T}$, and let $\sigma \subseteq \sigma'_v$ be the part of σ'_v that lies below p if λ' is a concave corner line w.r.t. $\mathscr{T}$. If $p \in \sigma'_h$ then let $\sigma \subseteq \sigma'_h$ be the part of σ'_h that lies to the right of p if λ' is a horizontal straight line or λ' is a convex corner line w.r.t. $\mathscr{T}$, and let $\sigma \subseteq \sigma'_h$ be the part of σ'_h that lies to the left of p if λ' is a concave corner line w.r.t. $\mathscr{T}$. Notice that in all cases σ is a bar line between p and a boundary point of λ'. Hence we can convert λ'' into a corner line in $\mathscr{P}$ by adding the point p and the line σ to it.

If there are at most two virtual corner lines that make up the set L' then there can be at most four straight lines that have to be converted to corner lines in $\mathscr{P}$: one for each horizontal and vertical part of the virtual corner lines. Lemma 3.25 states that the virtual corner lines in L' have corners that either have the same x- or y-coordinate. This means that the straight lines on either the vertical parts or the horizontal parts overlap. Hence if there are three virtual corner lines then two of each overlapping triple can be removed so that the resulting set of curves still is an a'-cut and the cut size decreases. Thus also in this case there are at most four straight lines in L' that have to be converted to corner lines in $\mathscr{P}$: three in either the horizontal or the vertical parts of the virtual corner lines and one in the other part. Therefore after converting L' and adding these curves to L, the resulting set of curves M' has a cut size of at most $5C + (6\sqrt{7/\varepsilon} + 2)\sqrt{a}$.

Notice that M', apart from λ, only contains straight and corner lines. Hence using Lemma 3.22 we can replace the staircase line λ with a set of corner and straight lines, yielding the set M. It only contains straight and corner lines and has a cut size of at most that of M'. What remains to be shown, in case the boundary of $\mathscr{T}$ does not contain λ, is that M cuts out an area of the desired size and that its cut size is of the desired length. The

set M cuts out an area of size m' where $m' \in [m - \varepsilon a, m + \varepsilon a]$. Since $\mathscr{T}$ is a tail that is not small, if $\mathscr{T} \subseteq \mathscr{A}(L)$ we can conclude that the size of $\mathscr{T}$ is greater or equal to a and hence $m \geq a$. If $\mathscr{T} \subseteq \mathscr{B}(L)$, since the surplus is part of $\mathscr{A}(L)$ obviously $m \geq a$ also holds in this case. Thus $m' \in [(1-\varepsilon)m, (1+\varepsilon)m]$, which establishes the desired size for the area. Concerning the cut size, let $\mathscr{R}$ be the rectangle that is defined by the boundary points of λ, let h be its height, w be its width, and let w.l.o.g. $h \geq w$. The length l_λ of λ is $l_\lambda = h + w > h$. Since both the deficit and the surplus of λ are contained in $\mathscr{R}$ we know that $a < hw \leq h^2$. Hence we can conclude that $C \geq l_\lambda > \sqrt{a}$. This means that the cut size of M is at most $(6\sqrt{7/\varepsilon} + 7)C$, as claimed.

Now consider the case when there is no tail cut out by L such that its curve is different from λ. This means that the only curve in L is λ. In this case we need to proceed differently than in the case before by reversing the transformation of the m-cut: we first remove λ and instead add the curves that, apart from λ, are contained in the boundary of λ's deficit. This yields a $(m+a)$-cut M' which contains only straight and corner lines, where a is the size of the deficit. Furthermore, if we assume w.l.o.g. that λ is oriented down and its deficit lies to the lower left side of λ, the corner lines all point up and right and are convex w.r.t. $\mathscr{A}(M')$. Since the deficit of λ is part of $\mathscr{A}(M')$, we can use Lemma 3.25 to find a set of straight and corner lines L' in $\mathscr{A}(M')$ that cuts out an area of size $a' \in [(1-\varepsilon)a, (1+\varepsilon)a]$. Again we need to convert those curves in L' that have a boundary point on one of the curves in M' into feasible curves in $\mathscr{P}$. Since the corner lines in both L' and M' point up and right, any curve in L' that has a boundary point in $\mathscr{A}(M')$ on one of the curves in M' can only have one such boundary point. Hence the same arguments as given above for the other case also apply for each such case here. We can thus make the necessary conversions of the curves in L', add the curves in M', and thereby yield the set of curves M which only contains straight and corner lines. As above it cuts out an area of size $m' \in [(1-\varepsilon)m, (1+\varepsilon)m]$, and has a cut size of at most $(6\sqrt{7/\varepsilon} + 7)C$, which concludes the proof. □

3.2.5 Converting Curves in Polygons to Segments in Grids

We have learnt that for any desired area m to be cut out from a simple polygon there exists a cut of only straight and corner lines that (1) cuts out at most a small amount $\varepsilon \cdot m$ more (or less) than the desired area, and (2) has a cut size that is close to the optimum (of arbitrary shape for area m). We summarise these results in the following corollary.

Corollary 3.29. *Let C be the cut size of an optimal m-cut L in some polygon $\mathscr{P}$. For any $\varepsilon \in]0,1]$ there exists a non-crossing corner m'-cut for some $m' \in [(1-\varepsilon)m, (1+\varepsilon)m]$, which has a cut size of at most $(54\sqrt{7/\varepsilon} + 63) \cdot C$.*

Proof. According to Corollary 3.15 we can assume that L only contains straight and corner lines except at most one curve which can either be a staircase or a rectangular line. In case L only contains straight and corner lines there is nothing to prove. In case it contains a staircase line the claim holds according to Theorem 3.28. In case L contains a rectangular line we can use Theorem 3.24 to convert L into an m-cut L'. It has a cut size of at most $9C$ and contains only straight and corner lines except at most one staircase line. If L' does not contain a staircase line, the claim obviously holds. Otherwise, using Theorem 3.28 on L', we can convert L into a non-crossing corner m'-cut, for some $m' \in [(1-\varepsilon)m, (1+\varepsilon)m]$, having a cut size of at most $(6\sqrt{7/\varepsilon} + 7) \cdot 9C$, which concludes the proof. □

Because our real interest is in cuts in grids, we now face the task to find a cut in the grid G given a cut in the polygon $\mathscr{P}_G$ constructed from G. Our transformation from a grid to a polygon implies that an optimal m-cut in G transforms into an m-cut in $\mathscr{P}_G$. But not necessarily into an optimal one, since the cut curves in the polygon are not limited to integer positions (these are integer positions in the dual of the grid, and thus halfway positions between grid points). In other words, a cut in the polygon does generally not translate directly into a cut in the grid (note that if we would just cut grid edges with polygon

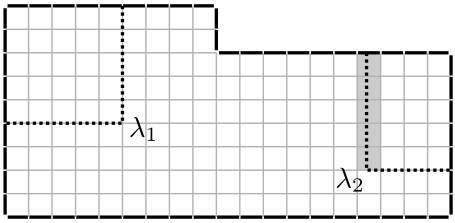


Figure 3.19: *A grid line λ_1 and a non-grid line λ_2. The corridor of λ_2 is shaded in grey. The boundary of the polygon is divided into lines of unit length.*

cut curves, that is, not cut them in the middle, this would not translate the cut out area into the same number of grid vertices). Whenever a curve in the cut of $\mathscr{P}_G$ happens to lie in integer position however, we will just take the corresponding segment to cut the grid G (Figure 3.19).

Definition 3.30 (corridor, grid line)**.** Given a grid $G = (V, E)$ let $\mathscr{S}_v$ be the axis-parallel unit square that has $v \in V$ as its centre and let γ_v be the boundary of $\mathscr{S}_v$. We consider a unit square to be an open set, i.e. $\gamma_v \cap \mathscr{S}_v = \emptyset$ for all $v \in V$. For any curve λ in $\mathscr{P}_G$ we refer to the set $\mathscr{C}_\lambda = \{p \in \mathscr{P}_G \mid \exists v \in V : \lambda \in \mathscr{S}_v \wedge p \in \mathscr{S}_v \cup \gamma_v\}$ as the *corridor* of λ. It is the union over the unit squares that intersect with λ together with their boundaries that are not part of the boundary of $\mathscr{P}_G$. A curve λ in $\mathscr{P}_G$ is called a *grid line* if $\mathscr{C}_\lambda = \emptyset$, i.e. λ lies on the boundaries of the unit squares.

For non-grid lines, we start with a clean-up phase that modifies a pair of these curves so that one of them becomes a grid line, and the other compensates for the area difference that this creates. We start the clean-up phase by first focussing on unit length open intervals on the polygon boundary between adjacent integer positions, as defined next.

Definition 3.31 (U_G)**.** Given a grid G let β be the boundary of $\mathscr{P}_G$. We define $\mathbb{H} = \left\{x - \frac{1}{2} \mid x \in \mathbb{N}\right\}$ so that $\mathbb{H}^2$ denotes the points between integer positions in the plane. Let the set U_G

contain all unit length curves in $\beta \setminus \mathbb{H}^2$.

Because a grid line λ does not hit any such open unit interval we are concerned only with cut curves that do. For any open unit interval hit by more than one cut curve, we can shift one of these curves to the boundary and compensate for the area difference by also shifting one other of these curves accordingly. Repeating this leaves us with at most one cut curve per open unit interval on the boundary.

Lemma 3.32. *For any grid G and any non-crossing corner m-cut L of cut size C in $\mathscr{P}_G$, there is a non-crossing corner m-cut M of cut size at most C in $\mathscr{P}_G$ such that there is no curve in U_G which includes more than one boundary point of curves in M.*

Proof. Consider the case when there is a curve $\delta \in U_G$ such that at least two curves in L have boundary points on δ. Without loss of generality let δ be the lower side of a unit square $\mathscr{S}_v$, i.e. the curves having a boundary point on δ lie above it. This means that any corner line having a boundary point on δ points down, while any such straight line is vertical. Let $K \subseteq L$ be the set of curves that have a boundary point on δ (Figure 3.20). We define (x_δ, y_δ) to be the lower left corner of the unit square $\mathscr{S}_v$ to which δ is the lower side. Let for any curve $\lambda \in K$ the point (x_λ, y_λ) be either the corner of λ, if it is a corner line, or the boundary point of λ that does not lie on δ, if λ is a straight line. We define

$$\mathscr{D}(\lambda) = \{(x, y) \in \mathscr{C}_\lambda \mid x \in]x_\delta, x_\lambda[\wedge\ y \in]y_\delta, y_\lambda[\ \}$$

to be the open set of points in λ's corridor that lie to the left of λ.

Since the curves in L are non-crossing, observe that if $\lambda \in K$ is a corner line pointing down and left any curve $\lambda' \in L$ that intersects $\mathscr{D}(\lambda)$, i.e. $\lambda' \cap \mathscr{D}(\lambda) \neq \emptyset$, must be a corner line and it must have the same orientation as λ. Thus λ' must also have a boundary point on δ since the lower boundary of $\mathscr{D}(\lambda)$ is part of δ. From this we can conclude that for a boundary point p on δ that belongs to a corner line pointing down and left, the

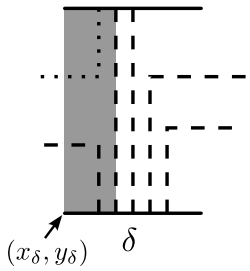


Figure 3.20: *The considered curve δ and the lines λ_1 to $\lambda_{|K|}$ from left to right (dashed). The area shaded in grey is $\mathscr{D}(\lambda_2)$.*

boundary points to the left of p on δ all belong to corner lines of the same orientation. Furthermore they must all be of smaller vertical length since the height of $\mathscr{D}(\lambda)$ equals the vertical length of λ. An analogous observation can be made for a corner line $\lambda \in K$ pointing down and right, if we consider the open set of points in its corridor that lie to the right of λ. Hence we can order the curves in K by traversing their boundary points on δ from left to right such that we first encounter corner lines pointing down and left with increasing vertical length, then straight lines, and finally corner lines pointing down and right with decreasing vertical length. Obviously this is also possible if some of the curves in K share the same boundary point on δ. Let the indices of the curves in $K = \{\lambda_1, ..., \lambda_{|K|}\}$ denote their position in this order (cf. Figure 3.20).

We will consider the curves λ_1 and λ_2 from K and in each case attempt to move the vertical part of λ_1 to the left until it intersects with the boundary of the unit squares, i.e. until the vertical line is a grid line. Thereafter we will find one or several curves that substitute λ_2 such that the resulting set of curves is an m-cut again. Towards this end, for any point (x, y) on the boundary of $\mathscr{P}_G$ such that $x \leq x_{\lambda_1}$ and $y \in [y_\delta, y_{\lambda_1}]$, we define the set of curves $\Lambda_1(x, y)$ as those straight and corner lines between (x, y) and (x_δ, y_δ) (Figure 3.21). That is $\mu \in \Lambda_1(x, y)$

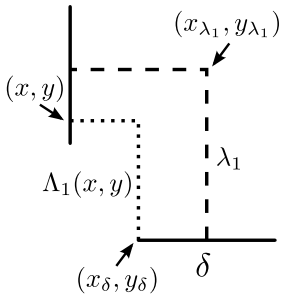


Figure 3.21: *The line λ_1 (dashed), a point (x, y) on the boundary of $\mathscr{P}_G$, and the corresponding set of curves $\Lambda_1(x, y)$ (dotted).*

if and only if μ is a segment curve and

$$\mu \subseteq \{(x', y') \in \mathscr{P}_G \mid (x' = x_\delta \wedge y' \in [y_\delta, y]) \vee (y' = y \wedge x' \in [x, x_\delta])\}.$$

Note that $\Lambda_1(x, y)$ contains more than one curve if $\mathscr{D}(\lambda_1)$ touches the boundary of $\mathscr{P}_G$ to its left.

Consider the case when λ_1 and λ_2 are corner lines pointing down and left (Figure 3.22). Let $(x, y) \notin \delta$ be the boundary point of λ_1 that does not lie on δ. We know that no curve from L intersects $\mathscr{D}(\lambda_1)$ by the observations made above. This means that removing λ_1 and adding the curves in $\Lambda_1(x, y)$ yields a non-crossing m'-cut for some m'. The difference between m and m' is equal to the size of $\mathscr{D}(\lambda_1)$ and is hence less than the size of $\mathscr{D}(\lambda_2)$. Also the only curve in L that intersects $\mathscr{D}(\lambda_2)$ is λ_1. Hence, after replacing λ_1, we can find a corner line μ pointing down and left that has its corner on λ_2's horizontal bar line and a boundary point on δ, such that removing λ_2 and introducing μ will again result in a non-crossing m-cut. Notice that the total length of the curves in $\Lambda_1(x, y)$ is shorter than the length of λ_1 and also the length of μ is shorter than the length of λ_2. Hence we obtain an m-cut of smaller cut size than C. Also the number of boundary points on δ is reduced by one.

In case the curves $\lambda_{|K|}$ and $\lambda_{|K|-1}$ are corner lines pointing

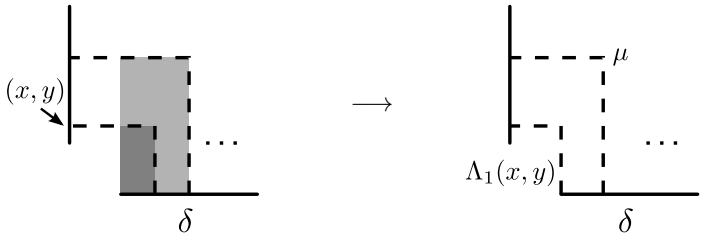


Figure 3.22: *The case when λ_1 and λ_2 are corner lines (left). They are substituted with $\Lambda_1(x, y)$ and μ (right). The area shaded in dark grey is $\mathscr{D}(\lambda_1)$, and the area shaded in both light and dark grey is $\mathscr{D}(\lambda_2)$.*

down and right we can use an analogous argumentation as the one given above to obtain an m-cut of cut size smaller than C. In the new cut the number of boundary points on δ is reduced by one. By repeating this procedure, we can transform L into an m-cut of smaller cut size. This can be done until there are at most two corner lines with boundary points on δ, such that they point down and left, and down and right, respectively. We thus assume in the remainder of the proof that K contains at most one such corner line each, while all others are straight lines.

Consider the case when λ_1 is a corner line pointing down and left and λ_2 is a straight line (Figure 3.23). In case the boundary points of λ_1 and λ_2 are the same on δ these two curves overlap. If q denotes the corner of λ_1, clearly we can introduce the corner line pointing up and left, remove λ_1 and λ_2, and thereby obtain an m-cut of smaller cut size than C. The number of boundary points on δ will then be reduced by two. In case λ_1 and λ_2 do not share the same boundary point on δ, we can again replace λ_1 with the curves in $\Lambda_1(x, y)$, exactly as above, yielding a non-crossing m'-cut L'. Let $\mathscr{C} \in \{\mathscr{A}(L'), \mathscr{B}(L')\}$ be the part of the m'-cut for which $\mathscr{D}(\lambda_1) \subseteq \mathscr{C}$. We define $\mathscr{L} \subseteq \mathscr{D}(\lambda_2) \cap \mathscr{C}$ to be the connected area for which $\mathscr{D}(\lambda_1) \subseteq \mathscr{L}$. That is, $\mathscr{L}$ lies to the left of λ_2 in L'. Since λ_1 and λ_2 do not overlap, the size of $\mathscr{D}(\lambda_1)$ is smaller than the size of $\mathscr{L}$, i.e. $\mathscr{D}(\lambda_1) \subset \mathscr{L}$. Hence we can

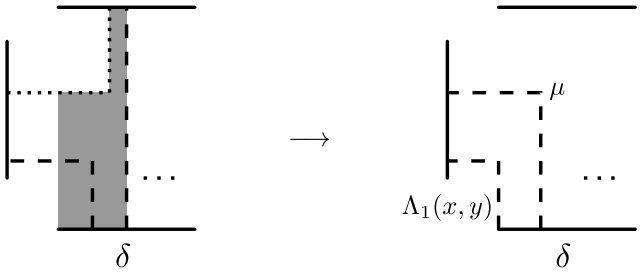


Figure 3.23: *The case when λ_1 is a corner line and λ_2 is a straight line (left). Together with λ' (dotted) they are substituted with $\Lambda_1(x, y)$ and μ (right). The area shaded in grey is $\mathscr{L}$.*

find a vertical straight line σ in $\mathscr{L}$ that cuts out an area the size of $\mathscr{D}(\lambda_1)$ on its right-hand side. One of the boundary points of σ (w.r.t. $\mathscr{L}$) lies on δ and the other boundary point can either lie on the boundary of $\mathscr{P}_G$ or on a curve $\lambda' \in L \setminus \{\lambda_1, \lambda_2\}$. In the former case we can replace λ_2 with σ and again yield an m-cut which has a smaller cut size and one boundary point less on δ. Otherwise, note that λ' lies in $\mathscr{D}(\lambda_2)$ and hence must be a corner line pointing up and left, since any other straight or corner line would either cross λ_1 or λ_2, or would have a boundary point on δ. This is not possible due to the choice of λ_2 in the ordering of K. This means that we can extend σ by a horizontal bar line σ' to a corner line $\mu = \sigma \cup \sigma'$ pointing down and left that has a corner on the horizontal bar line of λ'. Removing λ_2 and λ' and introducing μ instead will yield an m-cut with a smaller cut size and one boundary point less on δ.

Since we assumed that there is at most one corner line pointing down and right in K, if λ_1 is a straight line and λ_2 is such a corner line then $|K| = 2$. Hence this case is analogous to the case just covered.

Now consider the case when both λ_1 and λ_2 are straight lines (Figure 3.24). In case they overlap we can simply remove both lines. Otherwise it holds that $\mathscr{D}(\lambda_1) \subset \mathscr{D}(\lambda_2)$. As above, any curve from L that intersects $\mathscr{D}(\lambda_1)$ must be a corner line

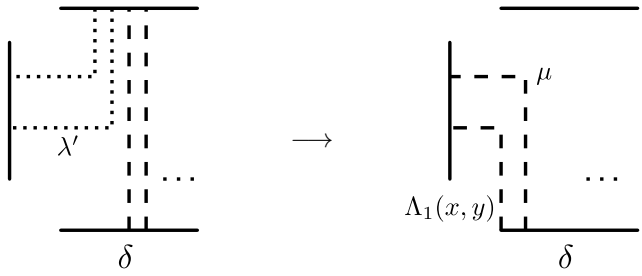


Figure 3.24: *The case when λ_1 and λ_2 are straight lines (left). Together with some curves in L' (dotted) they are substituted with $\Lambda_1(x, y)$ and μ (right).*

pointing up and left. Let $L' \subseteq L$ be the curves that intersect $\mathscr{D}(\lambda_1)$, and if $L' \neq \emptyset$ let $\lambda' \in L'$ be the one with the lowest and right-most corner among these. Notice that λ' is well-defined since the curves in L' are non-crossing. In this case we replace both λ_1 and λ' with $\Lambda_1(x, y)$, where (x, y) is the boundary point of the horizontal bar line of λ'. In case L' is empty we replace λ_1 with $\Lambda_1(x_\delta, y_{\lambda_1})$. In both cases we obtain an m'-cut for some m'. Analogous to the case when λ_1 is a corner line pointing down and left, and λ_2 is a straight line, we can find a curve with which to replace λ_2. As above we possibly also need to remove some other curve in L' to yield an m-cut of smaller cut size than C, and in which there is one boundary point less on δ.

The only case left is the one where both λ_1 and λ_2 are corner lines, i.e. the former points down and left, the latter down and right, and $|K| = 2$. We assume w.l.o.g. that the vertical length of λ_1 is at most that of λ_2. If λ_1 and λ_2 have the same boundary point on δ and they have the same vertical length, obviously we can remove these two curves and introduce a straight line that consists of the horizontal bars of λ_1 and λ_2 instead, and thereby obtain an m-cut with smaller cut size than C and with two boundary points less on δ. Consider the case when the two curves share the same boundary point on δ, they have different vertical lengths, and there is a corner line $\lambda' \in L$ pointing up and left having the same corner as λ_1. Then we can remove λ_1,

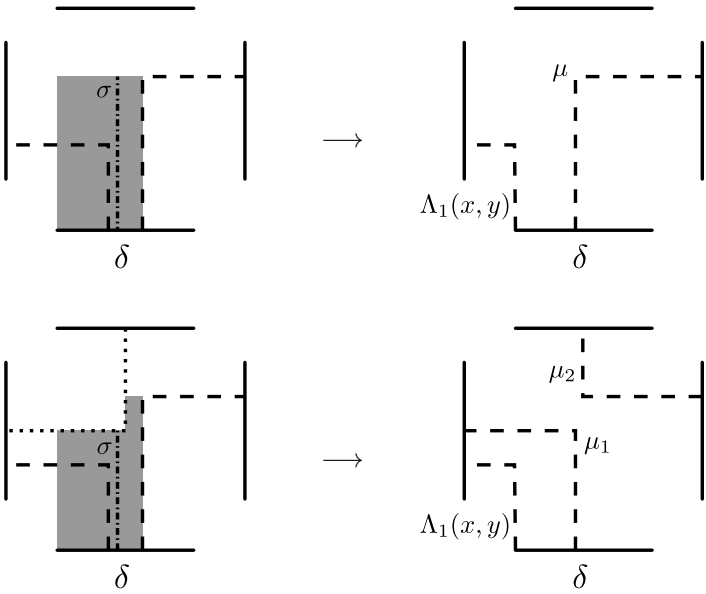


Figure 3.25: *The case when λ_1 and λ_2 are corner lines pointing in different directions (left). The area shaded in grey is $\mathscr{L}$. If σ (dashed and dotted) is the vertical bar line of μ, λ_1 and λ_2 can be substituted with $\Lambda_1(x, y)$ and μ (top). If σ ends at a line λ' (dotted), the lines λ_1, λ_2, and λ' can be substituted with $\Lambda_1(x, y)$, μ_1, and μ_2 (bottom).*

λ_2, and λ' and introduce the corner line pointing up and right that has the same corner as λ_2. We thereby yield an m-cut of smaller cut size in which two boundary points on δ are removed.

In all other cases let $(x, y) \notin \delta$ be the boundary point of λ_1 that does not lie on δ. Replacing λ_1 with $\Lambda_1(x, y)$ results in an m'-cut for some m', as in the case when λ_2 is a straight line (Figure 3.25). We define $\mathscr{L} \subseteq \mathscr{D}(\lambda_2)$ analogous to that case, i.e. $\mathscr{L}$ is the connected area to the left of λ_2 in the m'-cut. Furthermore let σ be the vertical straight line in $\mathscr{L}$ that cuts out an area the size of $\mathscr{D}(\lambda_1)$ on its right-hand side. Notice that $\sigma \subset \mathscr{L}$ is well-defined since we excluded all cases where

$\mathscr{D}(\lambda_1) = \mathscr{L}$. In case there is a corner line μ pointing down and right that has σ as its vertical bar line and overlaps with λ_2, we can replace λ_2 with μ and obtain an m-cut that has a cut size of at most C since the vertical length of λ_1 is at most that of λ_2. Also the number of boundary points on δ is reduced by one. Otherwise, similar to the case when λ_2 is a straight line, the boundary point of σ (w.r.t. $\mathscr{L}$) that does not lie on δ is either part of the boundary of $\mathscr{P}_G$, or it lies on a curve $\lambda' \in L$ that must be a corner line pointing up and left. In the former case we replace λ_2 with σ and yield an m-cut with the desired properties. In the latter case there are two corner lines μ_1 and μ_2 with the following properties. The first line points down and left, it has σ as its vertical bar line, and its horizontal bar line overlaps with λ'. The second line points up and right, has a vertical bar line that is part of λ', and a horizontal bar line that includes the horizontal bar line of λ_2. We can then replace λ_2 and λ' with μ_1 and μ_2 and yield an m-cut with cut size at most C, since the vertical length of λ_1 is at most that of λ_2. Also the number of boundary points on δ is reduced by one.

Notice that in all transformations above the number of boundary points on δ is reduced and at the same time the number of boundary points on other curves in U_G is never increased. We can hence repeat the above procedure for the curves in K and then in the same manner for all curves in U_G that include more than one boundary point of curves in the resulting m-cut. We yield an m-cut that has a smaller cut size than C, and for which any curve in U_G includes at most one boundary point of a curve in the m-cut. □

As long as there is more than one non-grid line (now in different open unit intervals on the boundary), we can shift one of them to become a grid line, and shift the other accordingly one to compensate for the area difference. This results in a situation with at most one non-grid line in the cut. During the whole process, the cut length does not increase, as the next lemma shows.

Lemma 3.33. *For any grid G and any non-crossing corner m-cut L of cut size C in $\mathscr{P}_G$, there is a non-crossing corner*

m-cut M of cut size at most C in $\mathscr{P}_G$ such that all curves in M except at most one are grid lines.

Proof. According to Lemma 3.32 we can assume that L contains no two curves that have boundary points that lie on the same curve from U_G. Let $K \subseteq L$ be the set of curves that are not grid lines and assume that $|K| \geq 2$. For a straight line $\lambda \in K$, any curve from K that intersects the corridor $\mathscr{C}_\lambda$ must have a boundary point on the same curve from U_G as λ. Hence no set of curves in L that have intersecting corridors include straight lines. For a corner line $\lambda \in K$, let $\mathscr{C} \in \{\mathscr{A}(\{\lambda\}), \mathscr{B}(\{\lambda\})\}$ be the area to which λ is convex and let $\overline{\mathscr{C}} = \{\mathscr{A}(\{\lambda\}), \mathscr{B}(\{\lambda\})\} \setminus \{\mathscr{C}\}$ be the area to which λ is concave. We define $\mathscr{V}(\lambda) = \mathscr{C}_\lambda \cap \mathscr{C}$ to be the part of λ's corridor to which λ is convex and $\overline{\mathscr{V}}(\lambda) = \mathscr{C}_\lambda \cap \overline{\mathscr{C}}$ to be the part to which λ is concave. Any curve in K that intersects $\mathscr{V}(\lambda)$ must have a boundary point on the same curve in U_G as λ. Therefore no curve in L intersects with $\mathscr{V}(\lambda)$.

We can thus conclude that if $\lambda_1, \lambda_2 \in K$ is a pair of curves with intersecting corridors, then λ_1 and λ_2 must be corner lines and λ_i, for $i \in \{1, 2\}$, must intersect $\overline{\mathscr{V}}(\lambda_j)$, where $j \in \{1, 2\} \setminus \{i\}$. Assume w.l.o.g. that λ_1 points down and left. Observe that this means that λ_2 points up and right and no other curve in K can intersect the corridors of λ_1 or λ_2, since otherwise there would be curves in L that have boundary points on the same curve from U_G. Notice that the corridors of λ_1 and λ_2 intersecting means that the corridors of the horizontal bar lines of λ_1 and λ_2 or the corridors of the corresponding vertical bar lines intersect. Assume w.l.o.g. that the vertical bar lines σ_1 and σ_2 of λ_1 and λ_2, respectively, are not grid lines and that the length l_1 of σ_1 is at most the length l_2 of σ_2. As in the proof of Lemma 3.32 we define $\mathscr{D}(\lambda_i)$, for both $i \in \{1, 2\}$, to be the open set of points to the left of λ_i, i.e. the height of $\mathscr{D}(\lambda_i)$ equals l_i and $\mathscr{D}(\lambda_1) \subseteq \mathscr{V}(\lambda_1)$ but $\mathscr{D}(\lambda_2) \subseteq \overline{\mathscr{V}}(\lambda_2)$. In this setting we assume w.l.o.g. that the size of $\mathscr{D}(\lambda_1)$ is at most the size of $\mathscr{D}(\lambda_2)$. We thus replace λ_1 with $\Lambda_1(x, y)$, as defined in Lemma 3.32, where (x, y) is the boundary point of the horizontal bar line of λ_1, and yield a non-crossing m'-cut for some m'. If $l_1 < l_2$ there exists a corner line λ_2' pointing up and right that contains the horizontal bar line of λ_2 and intersects $\mathscr{D}(\lambda_2)$, such that replacing λ_2 with λ_2'

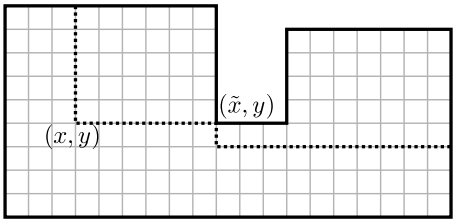


Figure 3.26: *A virtual pseudo-corner line with its corner at* (x, y) *and its unit sized step at* $\tilde{x}$*.*

yields a non-crossing m-cut. If $l_1 = l_2$ we can find an according virtual corner line that contains the horizontal bar line of λ_2 and intersects the boundary of $\mathscr{D}(\lambda_2)$. Notice that the new m-cut has a cut size of at most C since $l_1 \leq l_2$ and hence σ_1 was "moved" farther to the left than (or equally far as) σ_2. Also note that the new cut has at least one curve less in U_G containing a boundary point since the vertical bar lines in $\Lambda_1(x, y)$ are grid lines.

For any two curves in K with non-intersecting corridors we can use an analogous transformation as above. Since each transformation yields an m-cut in which there is at least one curve less in U_G with a boundary point, by repeating the above procedure we can transform L into the m-cut M with the desired properties. □

From now on, we can limit ourselves to the situation with only one non-grid line in the polygon cut. We shift this line to the nearest integer position (Figure 3.19), creating the need to compensate for the area difference. We do this by introducing more grid lines. But since this increases the cut length, we need to prove that the extra grid lines we introduce are short. In the end, this will preserve the property that the cut out area lies in the interval defined by m and ε, but will increase the cut size only by a small factor. Next, we will look at a way to cut out for compensation, and then argue that there is a place from which to cut out in this way.

We manage to compensate in a recursive manner. We compensate for an area difference a by first finding a particular way to cut out an area guaranteed to be between a and $3a/2$, with the exact value not under our control. This leaves us with the problem to compensate for at most half the previous area (since we are at most $a/2$ away from a). A recursive repetition of this compensation step ends after at most $\log(a)$ steps. The particular way to cut out the area between a and $3a/2$ makes use of a staircase grid line of three consecutive bends, with a step of unit height at the middle bend (Figure 3.26). Furthermore, the middle bend is guaranteed to lie outside or on the boundary of the polygon, so that the intersection of the staircase with the polygon results in a set of corner and straight lines in the cut. We call this a *virtual pseudo-corner line*. The analysis of the recursion reveals that the total length of the additional curves to cut out area a is limited to $3a$.

Definition 3.34 (virtual pseudo-corner line)**.** For any polygon $\mathscr{P}_G$ of a grid G a *virtual pseudo-corner line* is a set of grid lines Λ in $\mathscr{P}_G$ containing only straight and corner lines for which there are two points (x, y) and $(\tilde{x}, y-1)$, where $\tilde{x} \geq x$ and $(\tilde{x}, y) \notin \mathscr{P}_G$, such that $\lambda \in \Lambda$ if and only if

$$\begin{aligned}\lambda \subseteq &\{(x, y') \in \mathscr{P}_G \mid y' \geq y\} \cup \\ &\{(x', y) \in \mathscr{P}_G \mid x' \in [x, \tilde{x}]\} \cup \\ &\{(\tilde{x}, y') \in \mathscr{P}_G \mid y' \in [y-1, y]\} \cup \\ &\{(x', y-1) \in \mathscr{P}_G \mid x' \geq \tilde{x}\}.\end{aligned}$$

We call the unit step $\{(\tilde{x}, y') \in \mathbb{R}^2 \mid y' \in [y-1, y]\}$ the *break*, and (x, y) the *corner* of Λ. The length of Λ is the sum of the lengths of the included straight and corner lines. If Λ cuts out an area of size a on the upper right side of its corner, we say that it is a virtual pseudo-corner line *for* a.

A virtual pseudo-corner line is a special kind of virtual staircase line containing only grid lines. Notice that a virtual corner line containing only grid lines is a virtual pseudo-corner line. This is because the break of the latter can entirely lie outside of the polygon.

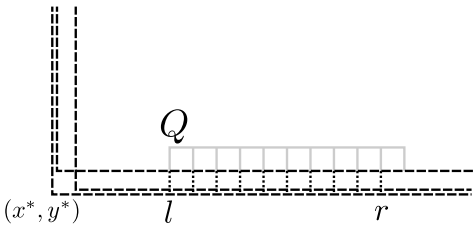


Figure 3.27: *The virtual corner line at (x^*, y^*) cuts out an area of size more than $\frac{3}{2}b$ while those at $(x^* + 1, y^*)$ and $(x^*, y^* + 1)$ cut out at most $b - 1$. If all points $(x^* + i, y^* + 1)$, where $i \in \{l, ..., r\}$, are inside the polygon, then all unit squares from Q are also inside. Since there are $r - l + 1 > b/2$ of them, the area $\mathscr{P}(x^* + 1, y^* + 1)$ has size greater than $b/2$.*

We first convince ourselves that the needed virtual pseudo-corner line exists. In case there is a virtual corner line that cuts out the required area and contains only grid lines we are done. In the other case a suitable set of curves can be constructed using three virtual corner lines at some integer points (x^*, y^*), $(x^* + 1, y^*)$, and $(x^*, y^* + 1)$ (Figure 3.27). These three virtual corner lines are chosen such that the first one cuts out an area larger than $3a/2$, while the other two each cut out at most $a - 1$. Using these properties it is then possible to show that there must be a unit sized step, i.e. a break, between the virtual corner lines at (x^*, y^*) and $(x^*, y^* + 1)$ with which a suitable virtual pseudo corner line can be constructed. That is, the corresponding set of curves cuts out an area between a and $3a/2$, and the upper most point of the break is on the boundary or outside of the polygon.

Lemma 3.35. *For any grid G with n vertices and any $b \in \{0, ..., n\}$, there is a value $a \in \left[b, \frac{3}{2}b\right]$ for which there exists a virtual pseudo-corner line Λ for a in $\mathscr{P}_G$.*

Proof. Since the vertices of the grid G are points with integer coordinates, i.e. $V \subset \mathbb{N}^2$, a virtual corner line contains only grid lines if its corner is a point in the set $\mathbb{H}^2$. If there is a virtual corner line for some $a \in \left[b, \frac{3}{2}b\right]$ with a corner in $\mathbb{H}^2$ then the

lemma holds. Assume no such virtual corner line exists. Since any set of grid lines cuts out an area of integer size, this means that any virtual corner line with a corner from $\mathbb{H}^2$ either cuts out an area of size at least $\lceil \frac{3}{2} \rceil b$ or at most $b-1$.

Let $\Lambda(p)$ denote the virtual corner line with corner $p \in \mathbb{H}^2$ and let $\mathscr{P}(p)$ denote the area cut out by $\Lambda(p)$ on the upper right side of p. Under the above assumption, clearly there must be a point $(x, y) \in \mathbb{H}^2$ such that the size of $\mathscr{P}(x, y)$ is greater than $\frac{3}{2}b$ since $b \leq n$, and obviously there is a point $(x', y') \in \mathbb{H}^2$ with $x' \geq x$ and $y' \geq y$ such that $\mathscr{P}(x', y') = \emptyset$. Because the area $\mathscr{P}(p)$ for any $p \in \mathbb{H}^2$ includes any area $\mathscr{P}(q)$ of a corner q above or to the right of p, the size of $\mathscr{P}(p)$ is monotonically decreasing in both coordinates of p. Hence we can find a point $(x^*, y^*) \in \mathbb{H}^2$ with $x^* \in [x, x']$ and $y^* \in [y, y']$ such that the size of $\mathscr{P}(x^*, y^*)$ is at least $\lceil \frac{3}{2} \rceil b$ while the size of both $\mathscr{P}(x^* + 1, y^*)$ and $\mathscr{P}(x^*, y^* + 1)$ are at most $b - 1$.

Let $\mathscr{P}_{ij} = \mathscr{P}(x^* + i, y^* + j)$ and $\Lambda_{ij} = \Lambda(x^* + i, y^* + j)$ for $i, j \in \mathbb{N}_0$. The area $\mathscr{P}_{00} \setminus \mathscr{P}_{01}$ has height 1 and contains a series of unit squares. For any $x \in \mathbb{N}$ the difference between the area $\mathscr{P}_{00}$ and $\mathscr{P}_{01} \cup \mathscr{P}_{x0}$ includes only unit squares from $\mathscr{P}_{00} \setminus \mathscr{P}_{01}$. Hence the above bounds on the sizes of $\mathscr{P}_{00}$ and $\mathscr{P}_{01}$ mean that we can find two integers $l, r \in \mathbb{N}$ such that $l \leq r$ and the size of $\mathscr{P}_{01} \cup \mathscr{P}_{l0}$ equals $\lfloor \frac{3}{2} b \rfloor$ and the size of $\mathscr{P}_{01} \cup \mathscr{P}_{r0}$ equals b. If for a value $i \in \{l, ..., r\}$ there is a pair of crossing curves in $\Lambda_{01} \cup \Lambda_{i0}$, their crossing point is $p_{i1} = (x^* + i, y^* + 1)$. If however there exists a corresponding value for i such that there are no curves in $\Lambda_{01} \cup \Lambda_{i0}$ that cross, then let Λ include the curves to the left of p_{i1} from the first set together with the curves below p_{i1} from the second set, i.e.

$$\Lambda = \{\lambda \in \Lambda_{01} \mid \forall (x, y) \in \lambda : x < x^* + i\} \cup \{\lambda \in \Lambda_{i0} \mid \forall (x, y) \in \lambda : y < y^* + 1\}. \quad (3.9)$$

Clearly the set Λ fulfils the lemma. Hence it remains to show that we can always find a corresponding value i such that p_{i1} is not in $\mathscr{P}_G$.

Assume this is not the case, i.e. for any value $i \in \{l, ..., r\}$ it holds that $p_{i1} \in \mathscr{P}_G$. This means that any unit square that has

one of these points as a corner must be included in $\mathscr{P}_G$. Let Q be the set of unit squares in $\mathscr{P}_{11}$ that have such a point p_{i1} as their lower left corner. There are $r - l + 1 \geq \lfloor \frac{3}{2}b \rfloor - b + 1 > \frac{1}{2}b$ many points p_{i1}. We can conclude that there are at least $\frac{1}{2}b$ many unit squares in Q. Since the squares have unit size and are included in $\mathscr{P}_{11}$ the size of $\mathscr{P}_{11}$ is at least $\frac{1}{2}b$.

Let us derive an upper bound on the size of the area $\mathscr{P}_{11} = \mathscr{P}_{10} \cap \mathscr{P}_{01}$. Since $\mathscr{P}_{10} \subseteq \mathscr{P}_{00}$ the size of $\mathscr{P}_{00} \setminus \mathscr{P}_{10}$ is at least $\frac{1}{2}b+1$. The difference between the area $\mathscr{P}_{00} \setminus \mathscr{P}_{10}$ and $\mathscr{P}_{01} \setminus \mathscr{P}_{10}$ can at most include the unit square $\mathscr{S}_v$ where $v = (x^* + \frac{1}{2}, y^* + \frac{1}{2})$. Whether $v \in V$ or not, this means that the size of the area $\mathscr{P}_{01} \setminus \mathscr{P}_{10}$ is at least $\frac{1}{2}b$. Since $\mathscr{P}_{01} \cap \mathscr{P}_{10} = \mathscr{P}_{01} \setminus (\mathscr{P}_{01} \setminus \mathscr{P}_{10})$, we can conclude that that the size of $\mathscr{P}_{11}$ is at most $\frac{1}{2}b - 1$. However this contradicts the lower bound derived above and hence the lemma holds. □

Using the above lemma we can show that an area of arbitrary size can be cut out recursively as described before.

Lemma 3.36. *For a grid G let Λ be a virtual corner line for b in $\mathscr{P}_G$ that contains only grid lines. If $\mathscr{P}$ denotes the area cut out by Λ on the upper right side of its corner, then for any $a \in \{0, ..., b\}$ there exists a set of non-crossing corner grid lines L in $\mathscr{P}_G$ cutting out an area of size a from $\mathscr{P}$. Furthermore, the curves in $\Lambda \cup L$ are non-crossing and the cut size of L is at most $3a$.*

Proof. Let $a_1 = a$ and G_1 be the grid corresponding to the area $\mathscr{P}$. Consider the following recursive procedure. In each step $i \geq 1$ we attempt to cut out an area of size a_i from $\mathscr{P}_{G_i}$ using only grid lines. According to Lemma 3.35 we can find a virtual pseudo-corner line Λ'_i in $\mathscr{P}_{G_i}$ for some $a'_i \in [a_i, \frac{3}{2}a_i]$ with the properties listed therein. We need to transform the curves in Λ'_i into a valid virtual pseudo-corner line Λ_i in $\mathscr{P}_G$ that cuts out the same area as Λ'_i. Assume for now that this can be done. We will describe the transformation later. If $a'_i = a_i$ the recursion terminates. Otherwise let $a_{i+1} = a'_i - a_i$ and let G_{i+1} be the grid that corresponds to the area $\mathscr{A}(\Lambda_i)$ of the a'_i-cut Λ_i, i.e. $\mathscr{P}_{G_{i+1}} = \mathscr{A}(\Lambda_i)$. From $a'_{i+1} \leq \frac{3}{2}a_{i+1} = \frac{3}{2}(a'_i - a_i)$ and $a_i \geq \frac{2}{3}a'_i$

we can conclude that $a'_{i+1} \leq \frac{1}{2}a'_i$, i.e. the area $\mathscr{A}(\Lambda_i)$ that is cut out from $\mathscr{P}_{G_i}$ is smaller than $\mathscr{P}_{G_i}$. If this procedure terminates the set $L = \bigcup_{i\geq 1} \Lambda_i$ clearly cuts out an area of size exactly a. Since any set Λ_i contains only grid lines, the size a'_i of the cut out area must be integer. By the fact that the cut out area in step $i+1$ has a size at most half the size of the cut out area in step i, this means that the procedure terminates after at most $\lfloor \log_2(a) \rfloor$ steps.

Since Λ_i contains only grid lines, the area $\mathscr{P}_{G_i}$ that is cut out by Λ_i can be decomposed into a'_i many unit squares. The set Λ_i contains at most two corner lines and therefore each, except at most two of the a'_i unit squares, has at most one side of its boundary coinciding with a curve in Λ_i. There may be two unit squares that each have two sides of their boundaries coincide with a corner line in Λ_i. Hence the length l_i of Λ_i is at most $a'_i + 2$. From $a'_{i+1} \leq \frac{1}{2}a'_i$ we can conclude that $a'_i \leq a/2^i$ which means that $l_i \leq a/2^i + 2$. Therefore the cut size C of L is

$$C = \sum_{i\geq 0} l_i \leq \sum_{i=1}^{\lfloor \log_2(a) \rfloor} \left(\frac{a}{2^i} + 2\right) \leq a\left(2 - \frac{2}{a}\right) + 2\log_2(a) \leq 3a,$$

where the last inequality holds since $2\log_2(a) - 2 < a$ for any $a > 0$. If $a = 0$ then $C = 0$ and the claimed bound still holds.

What remains to be shown is that we can convert the curve sets Λ'_i into valid virtual pseudo-corner lines Λ_i in $\mathscr{P}_G$. Since Λ is a virtual corner line containing only grid lines, it is also a virtual pseudo-corner line. We let $\Lambda_0 = \Lambda$ and then show by induction that each Λ'_i can be transformed into an appropriate Λ_i for $i \geq 1$. Assume that Λ_i is a virtual pseudo-corner line that cuts out the same area $\mathscr{A}(\Lambda_i)$ in $\mathscr{P}_G$ as Λ'_i does in $\mathscr{P}_{G_i}$. The set Λ'_{i+1} is an a'_{i+1}-cut in $\mathscr{P}_{G_{i+1}} = \mathscr{A}(\Lambda_i)$ that cuts out the area $\mathscr{A}(\Lambda'_{i+1}) \subseteq \mathscr{P}_{G_{i+1}}$. If β denotes the boundary of $\mathscr{P}_G$ and γ denotes the boundary of $\mathscr{A}(\Lambda'_{i+1})$, then we include all segment curves $\lambda \subseteq \gamma \setminus \beta$ in Λ_{i+1} and claim that it is a virtual pseudo-corner line. If it is then it clearly cuts out the same area as Λ'_{i+1}. The point set $\gamma \setminus \beta$ may contain parts of curves from Λ_i and Λ'_{i+1}. However, since these sets contain only grid lines and the length of a break is 1, $\gamma \setminus \beta$ can contain at most one

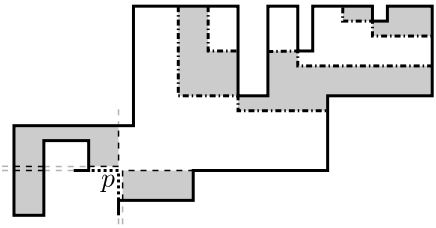


Figure 3.28: *A tail with its corner line at p (black dotted). The excess area in grey is cut out using the four virtual corner lines at p (thin dashed) together with the recursive method that uses virtual pseudo-corner lines (dashed and dotted).*

break from Λ_i and Λ'_{i+1}. It is easy to see that this means that Λ_{i+1} is a virtual pseudo-corner line. □

It remains to be shown that there is a place in the polygon to cut out from using the recursive method above. For this we use a tail of the cut (Figure 3.28), similar to the staircase line argument in the previous section. We have to make sure that there is a tail that is big enough to support an area of size a. For a non-crossing corner cut L containing only one curve λ that is not a grid line we call a tail $\mathscr{T} \subseteq \mathscr{A}(L)$ (respectively $\mathscr{T} \subseteq \mathscr{B}(L)$) *tiny* if the size of $\mathscr{T}$ is strictly smaller than the size of $\mathscr{C}_\lambda \cap \mathscr{B}(L)$ (respectively $\mathscr{C}_\lambda \cap \mathscr{A}(L)$). In the following we give a similar observation on such tails as was given for the case when they are small.

Lemma 3.37. *For a grid G, let L be a non-crossing corner m-cut in $\mathscr{P}_G$ with cut size C containing exactly one curve λ that is not a grid line. There exists a non-crossing corner m-cut M in $\mathscr{P}_G$ which contains exactly one curve that is not a grid line, has cut size of at most $C+1$, and the curve of any tiny tail cut out by M equals the curve that is not a grid line.*

Proof. The proof of this lemma is analogous to the proof of Lemma 3.27. However the non-grid line λ may get longer when the area of a tail is transferred to the corridor of λ. This can

only happen if λ is a corner line an then its length grows by at most 2. Since the curve of any tail is a grid line it has a length of at least 1. Hence the total cut size grows by at most 1. □

We need to make sure that no additional curves are produced while cutting out the area of size a from a tail which would increase the cut size by some non-constant factor. For this we break the tail into four sectors using four virtual corner lines having the same corner as the curve of the tail. We then greedily assign these virtual corner lines to the cut as long as the cut out area does not exceed a. The remaining difference to reach the desired area a is finally cut out using the recursive method presented above from one of the four sectors that was not yet used.

Lemma 3.38. *For a grid G, let L be a set of grid lines in the polygon $\mathscr{P}_G$ and let $\mathscr{T}$ be a tail cut out by L. If b denotes the size of $\mathscr{T}$, then for any $a \in \{0, ..., b\}$ there exists a set of non-crossing corner grid lines M in $\mathscr{P}_G$ cutting out an area of size a from $\mathscr{T}$ such that the curves in $M \cup L$ are non-crossing. Furthermore, the cut size of M is at most $3a$.*

Proof. Let $\lambda \in L$ be the curve of $\mathscr{T}$. If λ is a straight line then let p be one of its boundary points, and if λ is a corner line let p be its corner. There are four virtual corner lines in $\mathscr{T}$ having p as their corner, one for each possible orientation. These virtual corner lines Λ_1 to Λ_4 partition $\mathscr{T}$ into four (possibly empty) areas $\mathscr{T}_1$ to $\mathscr{T}_4$ such that $\mathscr{T}_i$ is cut out by Λ_i, where $i \in \{1, 2, 3, 4\}$, on the "convex side" of its corner. Let $I \subseteq \{1, 2, 3, 4\}$ be the set for which $i \in I$ if and only if $\mathscr{T}_i \neq \emptyset$. If a equals 0 or b then the lemma obviously holds. Assume that $0 < a < b$. There exists a (possibly empty) subset $J \subset I$ such that the size b_J of the union area $\mathscr{T}_J = \bigcup_{i \in J} \mathscr{T}_i$ is at most a while for any $j \in I \setminus J$ the size of the area $\mathscr{T}_J \cup \mathscr{T}_j$ is greater than a. For each $i \in J$ the set M contains the curves in Λ_i. Notice that λ can not be included in any of the sets Λ_i since the latter are virtual corner lines in the open set of points $\mathscr{T}$. Hence, in case the boundary of $\mathscr{T}_J$ includes λ we also include λ in M. So far these curves cut out an area of size b_J from $\mathscr{P}_G$.

Since all involved curves are grid lines, if b_i, for some $i \in I$, denotes the size of the area $\mathscr{T}_i$, we can decompose $\mathscr{T}_i$ into b_i many unit squares. The set Λ_i contains at most one corner line and therefore each except at most one of the b_i unit squares has at most one side of its boundary coinciding with a curve in Λ_i. There may be one unit square that has two coinciding sides of its boundary with the corner line in Λ_i. Hence the length l_i of Λ_i is at most $b_i + 1$ and therefore the cut size of $\bigcup_{i\in J} \Lambda_i$ is at most $b_J + |J|$. Note that the same bound holds for the curves included in M so far, even if $\lambda \in M$.

Let $j \in I \setminus J$. According to Lemmas 3.36 and 3.35 we can find a set M' of non-crossing corner grid lines in $\mathscr{P}_G$ that cut out an area of size $a - b_J$ from $\mathscr{T}_j$ such that the cut size of M' is at most $3(a - b_J)$. If we also include M' in M, we cut out an area of size a from $\mathscr{T}$ without crossing a curve in L. Furthermore the cut size of M is at most

$$b_J + |J| + 3(a - b_J) = 3a + |J| - 2b_J \leq 3a,$$

where the inequality holds since $\mathscr{T}_i \neq \emptyset$ and hence $b_i \geq 1$ for each $i \in J$. Thus the set M fulfils the required properties. □

The main result as stated in Theorem 3.2, follows from the next theorem which summarises the results of this section.

Theorem 3.39. *Let C be the cut size of an optimal m-cut L, for some $m \in \mathbb{N}$, in the polygon $\mathscr{P}_G$ of a grid G. For any $\varepsilon \in]0, 1]$ there exists a non-crossing corner m'-cut L' for some $m' \in [(1-\varepsilon)m, (1+\varepsilon)m]$, such that all curves in L' are grid lines and the cut size is at most $(216\sqrt{7/\varepsilon} + 261) \cdot C$.*

Proof. We can apply Corollary 3.29 and Lemma 3.33 to L, i.e. we know that there exists a non-crossing corner m'-cut M, for some $m' \in [(1-\varepsilon)m, (1+\varepsilon)m]$, with cut size at most $(54\sqrt{7/\varepsilon}+63)\cdot C$ in $\mathscr{P}_G$ such that M contains at most one curve that is not a grid line. If all curves in M are grid lines, we are done. If not then let $\lambda \in M$ be the curve that is not a grid line. In case there exists a tail cut out by M such that λ is not its curve, let $\mathscr{T}$ be this tail and assume w.l.o.g. that $\mathscr{T} \subseteq \mathscr{A}(M)$. By Lemma 3.37 we can assume that the size of $\mathscr{T}$ is at least the

size of $\mathscr{C}_\lambda \cap \mathscr{B}(M)$ if we allow the cut size of M to be at most $(54\sqrt{7/\varepsilon} + 63) \cdot C + 1$. Notice that, if a denotes the size of $\mathscr{C}_\lambda \cap \mathscr{B}(M)$, a is not necessarily an integer since m' might not be a natural number. However we can conclude that the size of $\mathscr{T}$ is at least $\lceil a \rceil$ since the curve of $\mathscr{T}$ is a grid line and hence $\mathscr{T}$ is of integer size. Let β denote the boundary of $\mathscr{P}_G$ and γ the boundary of $\mathscr{C}_\lambda \cap \mathscr{B}(M)$. In case λ is a corner line and contains a bar line σ that is a grid line, let Λ contain all straight and corner lines that are contained in the set $(\gamma \setminus (\beta \cup \lambda)) \cup \sigma$. In any other case let Λ contain the straight and corner lines in the set $\gamma \setminus (\beta \cup \lambda)$. By replacing the curve λ with the curves in Λ we yield an m''-cut M' where $m'' = m' + a$.

We attempt to cut out the excess area of size a in $\mathscr{T}$ using only grid lines. Notice that m'' must be an integer since M' contains only grid lines. If we assume w.l.o.g. that $m' \geq m$, since m is also an integer this means that $m'' - \lceil a \rceil$ is a natural number in the interval $[m, m']$. The latter is contained in $[(1-\varepsilon)m, (1+\varepsilon)m]$. Using Lemma 3.38 we can find a set of grid lines M'' that cut out an area of size $\lceil a \rceil$ from $\mathscr{T}$. The union $M' \cup M''$ forms a non-crossing set of grid lines cutting out an area from the interval $[(1-\varepsilon)m, (1+\varepsilon)m]$. Hence it remains to show (for the case when λ is not the curve of $\mathscr{T}$) that the cut size of $L' = M' \cup M''$ is bounded from above as claimed in the theorem.

Since λ is a straight or corner line, the size of the corridor of λ is at most the length of λ plus 1. Since the length of λ is upper-bounded by the cut size C' of M we can conclude that $\lceil a \rceil \leq a + 1 \leq C' + 2$. By Lemma 3.38 this means that the cut size of M'' is upper-bounded by $3(C' + 2)$. Clearly the length of Λ can be at most the length of λ plus 2. Hence also the cut size of M' is at most the cut size of M plus 2. Therefore the cut size of L' is at most $C' + 2 + 3(C' + 2) \leq (216\sqrt{7/\varepsilon} + 252) \cdot C + 9$. Cutting out an integer sized area greater than zero (and smaller than n) from the polygon $\mathscr{P}_G$, i.e. a polygon constructed from unit squares, will need a cut size C of at least 1. In this case the latter bound on the cut size of L' can be upper-bounded by the claimed bound of the theorem. If none (or all) of the area is to be cut out from $\mathscr{P}_G$, the trivial empty cut obviously also fulfils the requirements of this theorem.

Now consider the case when there is no tail such that λ is not its curve. This can only mean that there are two tails which both have λ as their curve and λ is the only curve in M. Let $\mathscr{T}$ be the tail that corresponds to the area $\mathscr{A}(M)$. Replacing λ with Λ as before, we obtain an m''-cut M' for which $\mathscr{A}(M') = \mathscr{A}(M) \cup (\mathscr{C}_\lambda \cap \mathscr{B}(M))$. Hence the size of $\mathscr{A}(M')$ is at least a. Since Λ may contain more than one curve, $\mathscr{A}(M')$ might not be a tail. Nevertheless, proving an analogous statement as Lemma 3.38 for this case we can come to the same conclusions as above. This is due to the fact that Λ is a virtual corner line, which conclude the proof. □

3.3 Recursive Applications

This section is concerned with computing approximate solutions to the BISECTION and k-BALANCED PARTITIONING problems on solid grid graphs, using the results achieved in the previous section. As mentioned in the introduction to this chapter, an attempt was made in [22] to compute optimal corner cuts in order to find good approximations to the BISECTION problem. However this attempt failed since the asymptotic runtime of the resulting algorithm is the same as for computing the optimal bisection. Thus we deviate from this direct approach and present a different method using solutions to the SPARSEST CUT problem. We will show that we can improve on the known algorithms for this problem when considering solid grid graphs. This is true both when computing optimal and approximate solutions.

The (approximate) solutions to SPARSEST CUT can be used together with the techniques developed by Leighton and Rao [44] in order to compute approximations to the EDGE SEPARATOR problem. The authors also show how to use these cuts to compute bicriteria approximations to the BISECTION problem. The cuts can furthermore be applied using the methods by Simon and Teng [65] to yield bicriteria approximations to the k-BALANCED PARTITIONING problem. Since we present an algorithm for SPARSEST CUT that is faster than any known algorithm on solid grid graphs, applying the latter techniques also improves

on the runtime to solve the BISECTION and k-BALANCED PARTITIONING problems for these graphs. At the same time the approximation guarantees for general graphs carry over to the grids. For the BISECTION problem a near-balanced partition can be computed for any constant $\varepsilon > 0$ in time $\mathcal{O}(n^{1.5})$. The cut size of the solution approximates the optimum within $\alpha \in \mathcal{O}(1/\varepsilon^3)$. Solutions to the k-BALANCED PARTITIONING problem can be computed in time $\mathcal{O}(n^{1.5} \log k)$ where the partition deviates by a factor of 2 from being perfectly balanced. Here the cut size is approximated within $\alpha \in \mathcal{O}(\log k)$. This factor of α is worse than the constant factor achieved by applying the Klein-Plotkin-Rao Theorem [40] to spreading metrics [20]. However the latter algorithm needs $\widetilde{\mathcal{O}}(n^3)$ time or $\widetilde{\mathcal{O}}(n^2)$ expected time. This shows that we are able to trade the solution quality for faster runtimes.

3.3.1 An Overview of the Used Techniques

The *sparsity* of a cut S is the ratio between its cut size $C(S)$ and the product of the sizes of its respective cut out parts A and B. That is, it is defined as

$$\frac{C(S)}{|A| \cdot |B|} .$$

The *sparsest cut* is a cut that minimises the sparsity, and is also the optimum solution to the SPARSEST CUT problem. Hence, intuitively, the goal is to find a cut that uses as few edges per cut out vertex as possible. Our main contribution leading to improved algorithms for BISECTION and k-BALANCED PARTITIONING on solid grid graphs is to show how solutions to SPARSEST CUT can be computed fast on these graphs. A main ingredient is to recall from [55, 60] that a sparsest cut contains only a single segment in a planar graph. Therefore a sparsest cut can be computed in quadratic time on solid grid graphs due to Lemmas 2.5 and 2.10 (pages 19 and 30). This is because a sparsest cut is an optimal m-cut (Definition 2.3 page 16) and thus only $\mathcal{O}(n^2)$ segments need to be considered. These can also be enumerated in the same time. This observation already improves on the previously fastest known algorithm to compute

sparsest cuts for these graphs by Park and Phillips [55]. Their algorithm runs in time $\mathcal{O}(n^3 \log n)$ but is also more general than ours since it can compute the sparsest cut for any planar graph.

The above observation however does not suffice to improve the runtime of the known approximation algorithms on solid grids for the BISECTION and k-BALANCED PARTITIONING problems. The reason for this is that Park and Phillips [55] also give approximation algorithms for sparsest cuts in planar graphs. These can also be used together with the techniques of Leighton and Rao [44], respectively Simon and Teng [65]. In particular it is possible to compute an $\mathcal{O}(t)$ approximation to the SPARSEST CUT problem in $\mathcal{O}(n^{1+1/t} \log^3 n)$ time for any planar graph. If t is constant the resulting approximation ratios for the BISECTION and k-BALANCED PARTITIONING problems asymptotically are the same. Hence using such an approximation to SPARSEST CUT improves the running times compared to using the optimal solution, while the approximation factors do not change.

However we are able to compute constant approximations to SPARSEST CUT for solid grid graphs in $\mathcal{O}(n)$ time. For this we show that if the segments are restricted to a given family the sparsest cut also contains only a single segment. As we will see this means that for solid grid graphs the *sparsest corner cut*, i.e. the sparsest cut among those restricted to straight and corner segments, approximates the sparsest cut within a constant factor. Since there is only a linear number of straight and corner segments according to Lemma 2.10 which can also be enumerated in the same time, this approximate solution can be computed in linear time. This in turn improves on the previously fastest known (but more general) algorithm by Park and Phillips [55] to compute such solutions for solid grid graphs.

As mentioned above we harness these results together with the techniques developed by Leighton and Rao [44] in order to compute approximations to the EDGE SEPARATOR problem. These solutions can in turn be used to compute approximations to BISECTION and k-BALANCED PARTITIONING using the methods of Leighton and Rao [44], respectively Simon and Teng [65]. All of these methods work by recursively cutting a graph. We will describe them in the following sections where they are needed.

Note however, that simply applying cuts recursively on solid grid graphs will not necessarily yield a solution. This is because a subgraph of a solid grid graph is not necessarily solid. Hence an intermediate step in the recursion might not be well-defined if only solid grids are considered. We will therefore generalise our techniques to grid graphs in which every connected component is solid.

3.3.2 Sparsest Cuts

To show how solutions to the `SPARSEST CUT` problem can be found fast on solid grid graphs we show that such a cut only contains a single segment. This is true even if the segments are restricted to a given family $\mathcal{T}$. A similar observation as the one given in the following lemma was also shown in [55, 60].

Lemma 3.40. *Let $\mathcal{T}$ be a family of segments in a solid grid graph G. Among the sparsest non-crossing $\mathcal{T}$-restricted cuts in G there is one containing only a single segment.*

Proof. Let S be a non-crossing $\mathcal{T}$-restricted m-cut. If there is more than one segment in S then there must be more than one connected component in either the A-part or the B-part. Assume w.l.o.g. that the A-part consists of several connected components and let W be one of them. Let x be number of vertices in W, and let $T \subset S$ be the set of segments which together cut out W. Since S is non-crossing, the set T is well-defined. Using the fact that $m(n-m) = nm - m^2$ we observe that the sparsity of S is

$$\begin{aligned}\frac{C(S)}{nm-m^2} &= \frac{C(T)+C(S\setminus T)}{n(x+(m-x))-(x+(m-x))^2}\\ &\geq \frac{C(T)+C(S\setminus T)}{nx-x^2+n(m-x)-(m-x)^2}.\end{aligned}$$

This means that either the sparsity of T or the sparsity of $S \setminus T$ is at most the sparsity of S. By repeatedly using this argument on a sparsest $\mathcal{T}$-restricted cut we can always find one that contains only a single segment from $\mathcal{T}$. □

The above observation immediately gives the following theorem by letting $\mathcal{T}$ be the family of relevant segments $\mathcal{S}$ in a solid grid graph. This is due to Lemmas 2.5 and 2.10, since a sparsest cut is an optimal m-cut and hence only a quadratic number of segments have to be considered.

Theorem 3.41. *For any solid grid graph an optimal solution to the* `SPARSEST CUT` *problem can be computed in* $\mathcal{O}(n^2)$ *time.*

As mentioned previously, this theorem does not help to improve the runtime of the desired algorithms. However using the results from Section 3.2 we can show that in solid grid graphs it suffices to restrict our attention to the family $\mathcal{C}$ of straight and corner segments in order to find a good approximation to a sparsest cut. Theorem 3.2 (page 53) entails that a constant approximation to a sparsest cut can be found using segments from the family $\mathcal{C}$, as we show next. We will generalise from solid grid graphs to graphs for which every connected component is solid. This is necessary for the following recursive algorithms since they may produce such graphs in intermediate steps.

Corollary 3.42. *For any grid graph G for which the connected components are solid, there is a non-crossing corner cut S with a sparsity which approximates the sparsest cut S^* of G by a constant factor.*

Proof. If G is disconnected then clearly any sparsest cut has sparsity 0 and its A-part contains all vertices of some subset of the connected components. This cut is the empty set of segments which trivially is a corner cut. Otherwise the sparsest cut S^* is an optimal m-cut for some m. Hence Theorem 3.2 in particular means that for S^* there is a non-crossing corner m'-cut S, where $m' \in [(1-\varepsilon)m, (1+\varepsilon)m]$, with a cut size of $\mathcal{O}(1/\sqrt{\varepsilon}) \cdot C(S^*)$. To compare the sparsities of S^* and S we need to get a better understanding of the behaviour of the function $C(S)/(m'(n-m'))$ measuring the sparsity of S. The denominator of this term is a function in the size of the A-part of S and reaches a maximum at $n/2$ in the interval $[0, n]$. For smaller values than $n/2$ the function is increasing while for larger values it is decreasing. Hence when considering the interval $[(1-\varepsilon)m, (1+\varepsilon)m]$ the

smallest value of the denominator lies at one of the extreme points $(1-\varepsilon)m$ or $(1+\varepsilon)m$. It can easily be verified that the smallest value is reached at $(1-\varepsilon)m$ if and only if $m \leq n/2$. Since we can assume that the latter is true w.l.o.g., we can lower bound the denominator of the sparsity of S by

$$(1-\varepsilon)m \cdot (n-(1-\varepsilon)m) \geq (1-\varepsilon)m(n-m).$$

For the sparsity itself this means that

$$\frac{C(S)}{m'(n-m')} \in \mathcal{O}\left(\frac{C(S^*)}{\sqrt{\varepsilon}(1-\varepsilon)\cdot m(n-m)}\right).$$

According to Theorem 3.2 we can choose ε freely between 0 and 1. Hence we can set $\varepsilon = 1/3$ which is the value at which the above term depending on ε is minimal. This means that the sparsity of S is at most a constant factor worse than the sparsity of S^*. □

The above observations entail that a constant approximation to a sparsest cut can be found in linear time by enumerating the segments in $\mathcal{C}$. For this we need to set $\mathcal{T} = \mathcal{C}$ in Lemma 3.40 and observe that the size of $\mathcal{C}$ is $\mathcal{O}(n)$ by Lemma 2.10 (page 30). Hence we get the following theorem.

Theorem 3.43. *For any solid grid graph a constant approximation to the* `SPARSEST CUT` *problem can be computed in $\mathcal{O}(n)$ time.*

3.3.3 Edge Separators

We now turn to using the above facts in order to compute an approximation to the `EDGE SEPARATOR` problem in solid grid graphs. Formally, given a value $b \leq 1/2$ we call an m-cut for which $bn \leq m \leq (1-b)n$ a *b-separator*. An *optimal* b-separator is one that minimises the cut size. Recall that a solution to the `EDGE SEPARATOR` problem is a b-separator for some $b \leq 1/3$. We use the techniques developed by Leighton and Rao [44], for which the following theorem summarises its qualities.

Theorem 3.44 (follows from [44]). *Let $b' \leq 1/3$ and $b \leq 1/2$ be given such that $b' < b$. Let also* **A** *be an algorithm that computes a cut with a sparsity that is within a factor β of the minimum sparsity. There is an algorithm that recursively uses* **A** *to compute a b'-separator. Moreover its cut size is at most $\mathcal{O}(\beta/(b-b'))$ times the optimum of a b-separator.*

We will briefly describe how the above algorithm works in order to determine its runtime on solid grid graphs. It uses a greedy strategy by recursively using solutions to the SPARSEST CUT problem. The algorithm cuts along an approximate sparsest cut and continues this procedure in the larger one of the resulting parts. This is continued until the currently considered part P has size at most $(1-b')n$. Since $b' \leq 1/3$ it easily follows that the total size of all the smaller parts considered in the intermediate steps is at least $b'n$. Hence the desired b'-separator is the set of segments that cut out the last part P from the given graph.[7]

The approximation guarantee results from the fact that what remains of the optimal b-separator in any considered graph during the intermediate steps has a certain sparsity. This sparsity can be upper-bounded using the (total) cut size of the b-separator which is divided by a term depending on both b and b'. The latter term bounds the size of what is left of the parts of the b-separator in the currently considered graph when b'-separators are used to cut off parts from the input. This results in the term $b - b'$ in the approximation guarantee. Since the algorithm **A** in Theorem 3.44 computes a β approximation to the sparsest cut, the cut size of $\mathcal{O}(\beta/(b-b'))$ times the optimum can be concluded (see [44] for more details).

Next we will show how corner cuts can be applied to compute approximations to the EDGE SEPARATOR problem on solid grid graphs. However, as before, because of the algorithms in the following sections we need to generalise to grid graphs that contain solid connected components.

Theorem 3.45. *Let $b' \leq 1/3$ and $b \leq 1/2$ be given such that*

[7] The algorithm can actually compute an approximation to EDGE SEPARATOR for any graph. Hence in general the solution is not a set of segments but a set of edges that cut out the part P.

$b' < b$. For any grid graph G for which the connected components are solid, a b'-separator can be computed. Its cut size is at most $\mathcal{O}(1/(b-b'))$ times the optimum cut size C^* of a b-separator. The time needed is $\mathcal{O}(\min\{n^2, nC^*/(b-b')\})$.

Proof. For a disconnected grid graph for which the connected components are solid, a sparsest cut can be computed by taking an arbitrary connected component and declaring its vertices to be the A-part of the empty set of segments. By Lemma 3.40, for any (connected) solid grid graph we can compute a non-crossing corner cut with lowest sparsity by considering all segments in the family $\mathcal{C}$ of straight and corner segments. We pick the one with smallest sparsity from these. Corollary 3.42 says that in both cases the computed cut is a constant approximation of the sparsest cut in the given grid.

The algorithm given by Theorem 3.44 can use these approximate sparsest cuts in order to compute an approximation to an optimal b-separator for any grid graph for which the connected components are solid. This is true since the segments in $\mathcal{C}$ all end at the boundary of the given grid. Hence they do not cut out parts lying entirely inside the grid. This means that no recursion step of the algorithm can introduce a hole into the grids that are induced by the two parts cut out by the approximate sparsest cuts. Hence Corollary 3.42 can be applied in every one of the recursion steps. We use the approximate sparsest cut to compute a b'-separator using Theorem 3.44, where we set $\beta \in \mathcal{O}(1)$. The result is an approximation to an optimal b-separator within ratio $\mathcal{O}(1/(b-b'))$.

It remains to be shown that the runtime of this algorithm is $\mathcal{O}(\min\{n^2, nC^*/(b-b')\})$. If the given graph is disconnected, in its recursive procedure the algorithm given by Theorem 3.44 first uses the sparsest cuts that cut out the vertex sets of the connected components. Only when a connected graph is reached during the recursion it starts cutting approximate sparsest cuts using the segments in $\mathcal{C}$. For a disconnected graph the vertex set of a connected component can be found in a number of steps that is linear in the size of the component. Hence the time needed by the algorithm in all recursion steps until the currently

considered graph is connected is $\mathcal{O}(n)$. We will argue that the rest of the time spent by the algorithm on a connected solid grid is $\mathcal{O}(\min\{n^2, nC^*/(b-b')\})$. By Theorem 3.43 a constant approximation to a sparsest cut in a solid grid graph can be computed in linear time. In any of the recursion steps of the algorithm the computed approximation cuts at least one edge if the given graph is connected. This means that the time needed to compute the b'-separator is $\mathcal{O}(n^2)$ since there are $\mathcal{O}(n)$ edges. However the number of edges can also be upper-bounded by the approximation ratio. Hence the algorithm also cuts at most $\mathcal{O}(C^*/(b-b'))$ edges and thus the claimed runtime follows. □

By Theorem 2.7 (page 23) any number of vertices can be cut from a grid graph using at most $\mathcal{O}(\sqrt{n})$ edges. Hence the optimum b-separator has cut size $C^* \in \mathcal{O}(\sqrt{n})$. From this we can conclude the following observation.

Corollary 3.46. *Let b and b' be two constants such that $b' \leq 1/3$, $b \leq 1/2$, and $b' < b$. For any grid graph G for which the connected components are solid grid graphs a b'-separator can be computed in $\mathcal{O}(n^{1.5})$ time. Its cut size is at most a constant times the optimum cut size C^* of a b-separator.*

3.3.4 Bisections

In the following we will show how to compute a near-balanced bisection for solid grid graphs for any $\varepsilon > 0$. The resulting cut size approximates the optimum of a perfectly balanced solution within a factor of $\alpha \in \mathcal{O}(1/\varepsilon^3)$. We use the results obtained above together with the methods presented by Leighton and Rao [44], as summarised in the following theorem.

Theorem 3.47 (follows from [44])**.** *Let $b' < b \leq 1/2$ and $d = b - b'$. Let also an algorithm $\mathbf{A}$ be given that computes a $d/2$-separator for which the cut size is within a factor β of an optimum d-separator. There is an algorithm that recursively uses $\mathbf{A}$ to compute a b'-separator. Moreover its cut size is at most $\mathcal{O}(\beta/d^2)$ times the optimum of a b-separator.*

In the following we give a brief description of the above algorithm. First a threshold parameter C is defined which is set to 1 and doubles until a b'-separator is found. In each recursion step the algorithm computes a $d/2$-separator using $\mathbf{A}$ for the current graph if it contains at least $dn/2$ vertices. If there are fewer than $dn/2$ vertices the recursion stops. Note that we may use the algorithm given by Theorem 3.44 for $\mathbf{A}$ since $d/2 \leq 1/3$. If the cut size of this $d/2$-separator is at most C then the considered graph is cut and the recursion continues on each of the two cut out parts. After the recursion finishes, all the pieces cut out by this process are packed into two sets such that their size difference is minimised. This can be done using a standard dynamic program for the `SUBSET SUM` problem [12]. If the two resulting sets constitute a b'-separator, the algorithm stops with the latter as output. Otherwise the threshold C is doubled and the algorithm continues the recursion by cutting out pieces using $\mathbf{A}$ and the above rules. It can be shown that the algorithm terminates for some value $C \in \mathcal{O}(\beta C^*)$, where C^* is the cut size of an optimal b-separator (see [44] for more details).

The approximation ratio on the cut size results from the fact that each piece that the algorithm cuts from the given graph has size $\Omega(d^2 n)$. This is because a piece is only cut when it contains at least $dn/2$ vertices, and in that case an $d/2$-separator is used. Hence there are at most $\mathcal{O}(1/d^2)$ pieces that the algorithm produces. Since the cost of cutting out a piece is at most $C \in \mathcal{O}(\beta C^*)$, the approximation ratio of $\mathcal{O}(\beta/d^2)$ follows.

We next show how near-balanced solutions to the `BISECTION` problem can be computed fast for solid grid graphs using the above observations.

Theorem 3.48. *For any $\varepsilon > 0$ there is an algorithm that computes a partition of the n vertices of a solid grid graph into two sets of size at most $(1+\varepsilon)\lceil n/2 \rceil$ each. The resulting cut size is approximated within a factor of $\alpha \in \mathcal{O}(1/\varepsilon^3)$. Moreover the algorithm runs in time $\mathcal{O}(\min\{n^3, n^{1.5}/\varepsilon^3\})$.*

Proof. We can use the algorithm given by Theorem 3.45 as al-

gorithm **A** in Theorem 3.47. The output of **A** is the set of segments S cutting out the last part P encountered during its recursion. The algorithm **A** uses only straight and corner segments in the intermediate steps, which all end at the boundaries of the respective cut out parts. Therefore the grid graph induced by P does not have any holes if the input grid to **A** does not. On the other hand, the other part $\overline{P}$ cut out by S is the union of the smaller parts encountered by **A** during its intermediate steps. Since the grid graphs induced by these parts also do not have holes, neither does the grid graph induced by $\overline{P}$. Hence if the input to **A** does not have holes, the graphs induced by the parts cut out by S may be disconnected but their connected components are solid. This means that **A** can be used in the recursion of the algorithm given by Theorem 3.47 if the input is a solid grid graph.

It is easy to check that when setting $b = 1 - \frac{\lceil n/2 \rceil}{n}$ and $b' = 1 - (1+\varepsilon)\frac{\lceil n/2 \rceil}{n}$ in Theorem 3.47 we get an algorithm that returns a near-balanced bisection. This means that $d = \varepsilon\frac{\lceil n/2 \rceil}{n}$ and hence the resulting approximation ratio is $\mathcal{O}(\beta/\varepsilon^2)$. As a consequence the approximation factor on the cut size for solid grid graphs is $\mathcal{O}(1/\varepsilon^3)$ by Theorem 3.45.

To determine the runtime we first consider the total time needed to cut the graph using the $d/2$-separators. Each such cut is made using the algorithm **A** provided by Theorem 3.45. It needs $\mathcal{O}(n\widetilde{C}/\varepsilon)$ time to do so in each recursion step, where $\widetilde{C}$ is the optimum cut size of a d-separator. By Theorem 2.7 we can assume that $\widetilde{C} \in \mathcal{O}(\sqrt{n})$ since any number of vertices can be cut out from a grid using $\mathcal{O}(\sqrt{n})$ edges. Hence a recursion step of the algorithm needs $\mathcal{O}(n^{1.5}/\varepsilon)$ time. Alternatively we can also bound this time by $\mathcal{O}(n^2)$ due to Theorem 3.45. The number of recursion steps made is bounded by the number of pieces that are cut from the graph. It was noted before that these are $\mathcal{O}(1/\varepsilon^2)$ many. Alternatively we can also bound this number by $\mathcal{O}(n)$. Hence the total time spent to cut the graph into pieces using **A** is $\mathcal{O}(\min\{n^{1.5}/\varepsilon^3, n^3\})$.

Additionally the algorithm needs to solve the `SUBSET SUM` problem for every value C of the threshold parameter. This

needs $\mathcal{O}(\min\{n/\varepsilon^2, n^2\})$ time, since the corresponding dynamic program [12] needs to pack $\mathcal{O}(\min\{1/\varepsilon^2, n\})$ pieces having n possible sizes. As mentioned above, the algorithm terminates for some value $C \in \mathcal{O}(C^*/\varepsilon)$. Also the threshold parameter C cannot exceed the number of edges in the grid which is $\mathcal{O}(n)$. Since the threshold parameter is doubled in each iteration, the dynamic program to pack the pieces needs to be executed $\mathcal{O}(\log(C^*/\varepsilon))$, respectively $\mathcal{O}(\log n)$, times. Thus the total amount of time needed by the algorithm to pack the cut out pieces is the minimum of $\mathcal{O}(n/\varepsilon^2 \cdot \log(C^*/\varepsilon))$ and $\mathcal{O}(n^2 \log n)$. Hence the runtime of the algorithm is dominated by the cutting of the grid, which concludes the proof. □

The next observations immediately follows from the above theorem. In particular it means that an approximation to the BISECTION problem can be computed in the same amount of time as an approximate solution to EDGE SEPARATOR. This can be concluded from Corollary 3.46.

Corollary 3.49. *For any $\varepsilon > 0$ there is an algorithm that computes a partition of the n vertices of a solid grid graph into two sets of size at most $(1+\varepsilon)\lceil n/2 \rceil$ each. The resulting cut size is approximated within a factor of $\alpha \in \mathcal{O}(1/\varepsilon^3)$. Moreover the algorithm runs in time $\mathcal{O}(n^{1.5})$ if ε is constant.*

3.3.5 Balanced Partitions

We now turn to applying the obtained observations on approximating the EDGE SEPARATOR problem in solid grid graphs to k-BALANCED PARTITIONING. We show how to compute a partition which is off by a factor of 2 from being perfectly balanced. This allows us to find a fast algorithm that approximates the cut size within a factor of $\alpha \in \mathcal{O}(\log k)$. To achieve this we use the techniques developed by Simon and Teng [65] that can be applied to any graph. The following theorem summarises the qualities of this algorithm.

Theorem 3.50 (follows from [65])**.** *Let* **A** *be an algorithm that computes a* 1/6*-separator for which the cut size is within a*

factor β of an optimum 1/3-separator. There is an algorithm that recursively uses **A** *to compute a partition of the vertices into k sets such that each set has size at most $2\lceil n/k \rceil$. Moreover its cut size is at most $\mathcal{O}(\beta \log k)$ times the optimum of a perfectly balanced solution.*

In order to determine the runtime we will briefly describe this algorithm. It is based on recursively using a solution to the `EDGE SEPARATOR` problem on each of the two resulting subgraphs. This is done until each cut out subgraph is of size at most $2\lceil n/k \rceil$. In a second phase the algorithm repeatedly merges the two smallest pieces resulting from the cutting phase until only k sets remain. It is easy to see that these k sets all have size at most $2\lceil n/k \rceil$. The solution returned by the algorithm is then the partition containing the k sets.

The guarantee on the cut size stems from the fact that the number of edges cut is compared with the cut size of a perfectly balanced solution. Any part P that is cut during the cutting phase is of size greater than $2\lceil n/k \rceil$, while the parts of an optimal perfectly balanced partition $\mathcal{V}^*$ are of size at most $\lceil n/k \rceil$. It is easy to see that one can construct a 1/3-separator in P from the connected components cut out from P by $\mathcal{V}^*$. The algorithm **A** in Theorem 3.50 computes a β approximation to the optimal 1/3-separator in P. Consider the recursion tree given by the algorithm in Theorem 3.50 and its recursive computation steps using **A**. For each computation step there is a node associated with it in the tree. Clearly all graphs associated with the nodes of one level of the recursion tree, i.e. those graphs considered at the same recursion depth, are disjoint. Hence it is possible to amortise the number of edges cut on one level by β times the cut size of $\mathcal{V}^*$. Since **A** cuts each part using a 1/6-separator until they have a size proportional to n/k, the height of the recursion tree is $\mathcal{O}(\log k)$. Hence the total cut size is at most $\mathcal{O}(\beta \log k)$ times the optimum.

We show next how an approximate solution to the k-`BALANCED PARTITIONING` problem on solid grid graphs can be computed fast if the set sizes are allowed to deviate by a factor of 2 from being perfectly balanced.

Theorem 3.51. *There is an algorithm that computes a partition of the n vertices of a solid grid graph into k sets of size at most $2\lceil n/k \rceil$ each. The resulting cut size is within a factor of $\alpha \in \mathcal{O}(\log k)$ from the optimum cut size C^* of a perfectly balanced solution. Moreover the algorithm runs in time $\mathcal{O}(nC^* \log k)$.*

Proof. By the same observations given in the proof of Theorem 3.48 we can conclude that the algorithm given by Theorem 3.45 can be used as algorithm **A** in Theorem 3.50. The result of the algorithm is thus a partition of the vertex set with the claimed approximation guarantees.

It remains to analyse the runtime of the algorithm. Consider the recursion tree given by the algorithm and its recursive computation steps using **A**. Let n_i be the size of the grid graph G_i associated with node i of the recursion tree. If $\mathcal{V}^*$ denotes the optimal perfectly balanced partition in the given grid graph, then let C_i be the number of edges in G_i that cut out $\mathcal{V}^*$. As noted before, this value is an upper bound on the cut size of the optimal 1/3-separator in G_i. Hence by Theorem 3.45 the algorithm **A** needs $\mathcal{O}(n_i C_i)$ time in G_i. Since all grid graphs associated with the vertices of one level of the recursion tree are disjoint, we can amortise the runtime of a given level by $\mathcal{O}(nC^*)$. Because the height of the recursion tree is $\mathcal{O}(\log k)$, the runtime of the cutting phase of this algorithm is $\mathcal{O}(nC^* \log k)$.

In the second phase of the algorithm the resulting pieces are merged until only k remain. This can be done by first sorting the pieces by their sizes and then merging the two smallest ones repeatedly. This needs $\mathcal{O}(k \log k)$ time since there are only $\mathcal{O}(k)$ leaves of the recursion tree which correspond to the pieces that need to be merged. The reason is that any parent node i of a leaf corresponds to a grid graph G_i containing at least $2\lceil n/k \rceil$ vertices. Otherwise the algorithm would not have cut it. Since a 1/6-separator is used to cut G_i, the grid corresponding to the leaf node contains at least $\lceil n/k \rceil/3$ vertices. Hence there are at most $\frac{n}{\lceil n/k \rceil/3} \leq 3k$ leaves. □

By Theorem 2.7 any number of vertices can be cut out from a grid graph using at most $\mathcal{O}(\sqrt{n})$ edges. Using this observation

the runtime of the above algorithm can be estimated as follows.

Corollary 3.52. *There is an algorithm that computes a partition of the n vertices of a solid grid graph into k sets of size at most $2\lceil n/k \rceil$ each. The resulting cut size is within a factor of $\alpha \in \mathcal{O}(\log k)$ from the optimum cut size of a perfectly balanced solution. Moreover the algorithm runs in time $\mathcal{O}(n^{1.5} \log k)$.*

Proof. The proof is the same as for Theorem 3.51. The only difference is that Corollary 3.46 is used instead of Theorem 3.45 to provide the algorithm **A**. This means that the time needed for a node i in the recursion tree is $\mathcal{O}(n_i^{1.5})$. Using Jensen's inequality [58] it is possible to amortise the runtime for a level of the tree by $\mathcal{O}(n^{1.5})$. Hence the total runtime follows. □

3.4 Improving the Approximation Ratios

We have seen that optimal corner cuts are good approximations to optimal m-cuts in solid grid graphs, and we were able to devise fast approximation algorithms based on this fact. The techniques used were those by Leighton and Rao [44], and by Simon and Teng [65] to compute approximate solutions to the `BISECTION`, respectively the k-`BALANCED PARTITIONING` problem.

One remaining question is whether the approximation guarantee given in Section 3.2 for the corner cuts can be improved. In particular it is not clear whether the ε factor, by which the size of the cut out part deviates from the given value m, is necessary. Also the final constant given by Theorem 3.39, which has a value of at least 832, seems very large. The reason for this large value is that in many of the lemmas leading to the theorem, the cut size of the involved curves grow by a constant factor. This means that the resulting constant grows exponentially with the number of intermediate steps used by the proof. Hence an improvement on the guaranteed approximation ratio may be achievable with a more direct approach than the one chosen here. In particular the best lower bound we can provide to compare optimal corner

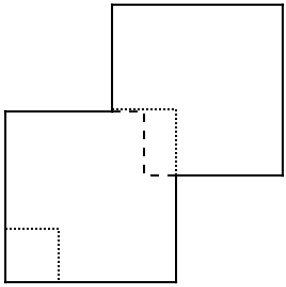


Figure 3.29: *A polygon in which the optimum corner bisection (dotted lines) has a cut size that is a factor of* $1 + 1/\sqrt{2}$ *larger than the optimum bisection (dashed line). Obviously this gives a lower bound for the corresponding grid graph.*

cuts with optimum m-cuts is $1 + 1/\sqrt{2}$ (Figure 3.29). Interestingly the lower bound example is a very simple one. There also exist more complicated examples based on the insights gained in this chapter. For instance it is possible to construct examples where the optimum corner bisection needs three segments. For this, topologies such as the one shown in Figure 3.11 can be used. However in all found examples the corner bisection with minimum cut size was also at most a factor $1 + 1/\sqrt{2}$ away from optimum.

Concerning the algorithms presented in Section 3.3, it remains open how to improve on the logarithmic approximation factor on the cut size for the k-`BALANCED PARTITIONING` problem on solid grid graphs when fast algorithms are desired. In particular since constant ratios are possible if slower runtimes are accepted (cf. Section 3.1.1). It is interesting to note that for the `BISECTION` problem we were able to devise an algorithm that computes a near-balanced solution for which the cut size increases the more stringent the limit on the set sizes is. However the runtime of the algorithm can be bounded independent of ε as shown by Theorem 3.48. Alternatively, algorithms which pay the price of computing near-balanced solutions not in the cut size but in the runtime could be conceivable.

For the k-BALANCED PARTITIONING problem the resulting partition deviates by a factor of 2 from being perfectly balanced while the runtime is fast, as shown by Corollary 3.52. For practical purposes it his however desirable to achieve a near-balanced partition. In particular this is true for data distribution applications in parallel-computing. If the load of a processor deviates by a factor of 2 from the perfectly balanced case, then this constitutes a significant slowdown of the whole computation. Hence an interesting question resulting from this chapter is whether near-balanced partitions can be computed for solid grid graphs. The next chapter will show that this is possible in polynomial time if ε is constant. The cut size of the computed solution approximates the optimum of a perfectly balanced solution by a logarithmic factor (independent of ε). However the runtime grows exponentially with decreasing values of ε. In the final chapter of this thesis we will show that this is unavoidable.

Chapter 4

Computing Near-Balanced Partitions

In this chapter we generalise to the edge-weighted version of the k-BALANCED PARTITIONING[1] problem in which the *cut cost*, i.e. the weighted cut size, needs to be minimised. We consider near-balanced[2] partitions for any given $\varepsilon > 0$. In the first part of this chapter we present an algorithm that solves the problem for weighted trees. The cut cost of the computed solution is at most the cut cost of an optimal perfectly balanced[2] partition. The algorithm runs in polynomial time if ε is constant and in this sense is a PTAS with respect to the balance of the partition. In the second part of this chapter we harness results on cut-based hierarchical graph decompositions into trees, in order to extend our PTAS for trees to general weighted graphs. The resulting algorithm approximates the cut cost within a logarithmic factor.

The results presented in this chapter were obtained in collaboration with Luca Foschini. They were published as an extended

[1] Definition 1.1 page 3
[2] Definition 3.1 page 51

abstract in [24] and will also partially appear in the doctoral thesis of Luca Foschini. Because of this, the details of the second part of this chapter (Section 4.3) are only sketched in this dissertation. The full proofs can be found in [24] and the thesis of Luca Foschini.

4.1 Cutting with the Balance in Mind

As mentioned in Chapter 3 the k-BALANCED PARTITIONING problem when $k = 2$, i.e. the BISECTION problem, is NP-hard [31]. Unfortunately, when k can be arbitrary it is even NP-hard [1] to compute any finite approximation when requiring perfectly balanced solutions. In order to overcome this obstacle, relaxing the balance constraints has proven beneficial. By that we mean that the sets of the partitions are allowed to be larger than $\lceil n/k \rceil$ by some given factor.

We have already seen an example of such an algorithm in Section 3.3.5 (page 128) which is based on the techniques developed by Simon and Teng [65]. This algorithm works in two phases of which the first cuts the graph into pieces and the second packs the pieces into k bins, i.e. the sets of the partition. This is a typical approach to solve the k-BALANCED PARTITIONING problem. Many of the strategies following this approach aim at breaking the graph into pieces of size at most $\lceil n/k \rceil$ while minimising the cut, only to later rely on the fact that pieces of that size can greedily be packed into k bins without their sizes exceeding $2\lceil n/k \rceil$. In this way the cutting phase can be oblivious of the packing phase. However it is not hard to imagine how a slack on the balance of this size can be detrimental to practical applications. In parallel-computing for instance, a ratio of two on the balance in the workload assigned to each machine can result in a slowdown by a factor of two. This is because the completion time is solely determined by the overloaded machines.

Therefore near-balanced partitions for which $\varepsilon > 0$ can be chosen arbitrarily, are of greater practical interest. However as ε approaches 0 and the constraint on the balance becomes more stringent, the cutting phase must break the given graph into

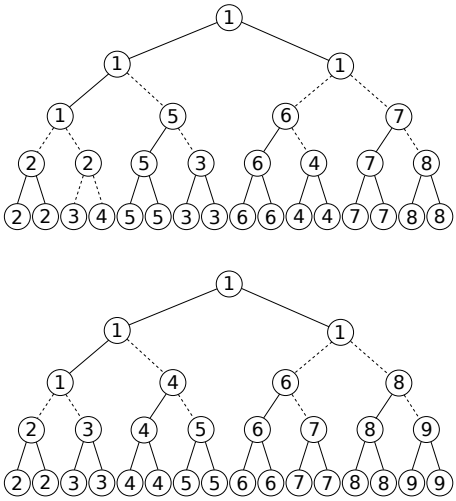


Figure 4.1: *Two unweighted binary trees that are optimally partitioned. For the tree on the top $k = 8$ (with a cut size of 10) whereas $k = 9$ (with a cut size of 8) for the tree on the bottom. The numbers in the vertices indicate the set they belong to and the cut edges are dashed.*

pieces more carefully so that they can later be packed into bins of the required size. One direct effect is that techniques that are oblivious of the packing phase while cutting the graph, do not extend to near-balanced partitions. Hence it is necessary to combine the packing with a cutting phase that already has the near-balanced solution in mind.

As argued above, the restriction to near-balanced partitions poses a major challenge to devising algorithms for k-`BALANCED PARTITIONING`. For this reason we consider very simple but non-trivial instances of the problem, namely trees. Figure 4.1 gives an example of how balanced partitions exhibit a counter-intuitive behaviour even on *perfect binary trees*, as increasing k does not necessarily entail a larger cut size. In the first part of this chapter we show that when near-balanced solutions are allowed, we are able to find substantially better solutions for trees than for general graphs. This is even true when the trees

are allowed to have positive edge weights. For edge-weighted graphs we measure the quality of the solution by its *cut cost* which is the sum of the weights of the edges connecting different sets of the partition. We present an algorithm for weighted trees that computes a near-balanced partition for any constant $\varepsilon > 0$ in polynomial time, achieving a cut cost no larger than the optimal of a perfectly balanced partition. This means that $\alpha = 1$ and hence the cut cost is not approximated. In this sense the presented algorithm is a PTAS w.r.t. the balance of the computed solution. In addition, our PTAS can be shown to yield an optimal *perfectly balanced* solution for trees if $k \in \Theta(n)$, while on general graphs the problem is NP-hard for these values of k [39].

In the second part of this chapter we capitalise on the PTAS for trees to tackle the k-`BALANCED PARTITIONING` problem on general graphs with positive edge weights. We use known techniques to decompose a graph into a set of trees [57]. By these results we can compute solutions for trees using our PTAS, and yield a near-balanced partition for the graph in which the cut cost is approximated within $\alpha \in \mathcal{O}(\log n)$. This is sufficient to simultaneously improve on the previous best result known [1] of $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$, and answer an open question posed in the same paper of whether an algorithm exists that has no dependence on ε in the approximation ratio α on the cut cost.

4.1.1 An Overview of the Used Techniques

Conceptually one could find a *perfectly balanced* partition of an edge-weighted tree T with minimum cut cost in two steps. First all the possible ways of cutting T into connected components are grouped into equivalence classes based on the sizes of their components. That is, the sets of connected components $\mathcal{W}$ and $\mathcal{W}'$ belong to the same equivalence class if they contain the same number of components of size x for all $x \in \{1, \ldots, \lceil n/k \rceil\}$. In a first step the set of connected components that achieves the cut of minimum cost for each class is computed and set to be the representative of the class. In a second stage only the equivalence classes whose elements can be packed into k

sets of size at most $\lceil n/k \rceil$ are considered, and among those the representative of the class with minimum cut cost is returned. Clearly such an algorithm finds the optimal solution to the k-BALANCED PARTITIONING problem, but the runtime is exponential in n as, in particular, the total number of equivalence classes is exponential. To get around this problem we instead group sets of connected components into coarser equivalence classes. These are determined by subdividing the possible component sizes into intervals. A coarse class then consists of cuts for which each interval in total contains the same number of component sizes. By making the lengths of the intervals appropriately depend on ε, this reduces the equivalence classes to a polynomial number if ε is constant. However this also introduces an approximation error in the balance of the solution.

We will show that for trees there is a dynamic program that computes the optimum way to cut a tree according to each given coarse equivalence class in polynomial time. That is, it computes the representatives of all coarse classes. In a second phase each of the computed representatives is packed into bins of size $(1+\varepsilon)\lceil n/k \rceil$. This is done using a known algorithm by Hochbaum and Shmoys [33] to approximate the BIN PACKING problem. The output of the algorithm is the packing among those that fit into k bins for which the smallest cut cost was produced in the first phase. The representative of the class to which also the perfectly balanced optimum belongs has a cut cost that is at most the optimum. Furthermore we will see that this representative is packed into k bins by the second phase. Therefore the output of the algorithm also has a cut cost that is at most the optimum, i.e. $\alpha = 1$.

We further harness the PTAS for trees by using it on *cut-based hierarchical decompositions* of general graphs. Informally such a decomposition of a graph G is a set of trees for which the leaves correspond to the vertices of G, and for which the cuts approximate the cuts in G. Since $\alpha = 1$ for the PTAS, the total approximation factor paid for the graphs is due only to the distortion of the cuts in the trees. That is, we yield an $\alpha \in \mathcal{O}(\log n)$ approximation on the cut cost when using the results from [57]. Note that since the graph is decomposed into

trees as a preliminary step, the decomposition is oblivious of the balance constraints related to solving k-BALANCED PARTITIONING on the individual trees. Therefore the cut distortion does not depend on ε.

4.1.2 Related Work

Andreev and Räcke [1] show that approximating the cut size of the k-BALANCED PARTITIONING problem is NP-hard for any finite factor on general graphs, if perfectly balanced partitions are desired. The authors also give a bicriteria approximation algorithm with $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$ when the solutions are allowed to be near-balanced for arbitrary $\varepsilon > 0$. The method recursively cuts along approximate solutions to the EDGE SEPARATOR problem. For this the technique by Leighton and Rao [44] is employed, which uses approximate solutions to the SPARSEST CUT problem (see Section 3.3.3 for a detailed description). The best algorithm known for general graphs [4] approximates the sparsest cut within a factor of $\mathcal{O}(\sqrt{\log n})$. In each recursion step of the algorithm by Andreev and Räcke, due to Theorem 3.44 (page 123), solutions to EDGE SEPARATOR with approximation ratio $\mathcal{O}(\sqrt{\log n}/\varepsilon)$ can be computed by cutting along $\varepsilon/2$-separators. The authors go on to show that their algorithm approximates the cut cost of the optimum perfectly balanced partition by the same ratio in each recursion step. They amortise this ratio for each level of the resulting recursion tree, and note that this tree has height $\mathcal{O}(\log n/\varepsilon)$. This gives the claimed approximation factor of $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$ for k-BALANCED PARTITIONING. We improve this result by giving an algorithm achieving $\alpha \in \mathcal{O}(\log n)$.

The following results can be used to compute partitions in which the set sizes are at most $2\lceil n/k \rceil$. Even *et al.* [20] present an algorithm using spreading metric techniques for which the cut cost is approximated within $\alpha \in \mathcal{O}(\log n)$. As reviewed in Chapter 3, Simon and Teng [65] conceived a technique that approximates the cut cost within $\alpha \in \mathcal{O}(\sqrt{\log n} \log k)$ using the best algorithm known [4] for the SPARSEST CUT problem. Later Krauthgamer *et al.* [42] improved both of these results to $\alpha \in \mathcal{O}(\sqrt{\log n \log k})$ using a semidefinite relaxation which

combines l_2^2 metrics with spreading metrics. For graphs with excluded minors (such as planar graphs) it is possible to apply a spreading metrics relaxation and the famous Klein-Plotkin-Rao Theorem [40] to compute solutions with a constant approximation factor on the cut cost. To compute perfectly balanced partitions, MacGregor [47] gives a greedy algorithm for trees that yields $\alpha \in \mathcal{O}(\Delta \log_\Delta(n/k))$, where Δ is the maximum degree of the tree. To the best of our knowledge the latter two are the only results for k-BALANCED PARTITIONING on restricted graph classes, apart from our own.

In addition to the BISECTION problem where $k = 2$ (as reviewed in Sections 2.1.2 and 3.1.1), some results are known for other extreme values of k. For trees the bisection algorithm by MacGregor [47] mentioned in Section 2.1.2 (page 14) is easily generalised to solve the k-BALANCED PARTITIONING problem for any constant k in polynomial time. At the other end of the spectrum, i.e. when $k \in \Theta(n)$, it is known that the problem is NP-hard [39] for any $k \leq n/3$ on general graphs. Interestingly the only (non-trivial) case when k-BALANCED PARTITIONING is not NP-hard on general graphs is when $k = n/2$. In this case the problem can easily be seen to be equivalent to the MAXIMUM MATCHING problem, in which a set of non-adjacent edges of maximum cardinality is to be found. It is well-known that this problem has polynomial time algorithms [29]. For $k \in \Theta(n)$ Feo and Khellaf [27] give an $\alpha = n/k$ approximation algorithm for k-BALANCED PARTITIONING, which was improved [26] to $\alpha = 2$ in case k equals $n/3$ or $n/4$.

We complete the review of related work by discussing the literature on cut-based hierarchical decompositions, which we leverage in our algorithm for general graphs. These decompositions have been studied in the context of oblivious routing schemes. Räcke [57] introduced an optimal decomposition with factor $\mathcal{O}(\log n)$, which we employ in the present work. More recently, Madry [48] showed that it is possible to generalise Räcke's insights to any *cut-based problem* (see [48] for more details). This result directly translates to our scenario and hence we use his notation in the present work.

4.2 A PTAS for Edge-Weighted Trees

In this section we present an algorithm that computes a near-balanced partition for any tree $T = (V, E, \omega)$ with positive edge weights $\omega : E \mapsto \mathbb{R}^+$. We will show that the cut cost of the obtained solution is at most the cut cost of the perfectly balanced optimum, i.e. $\alpha = 1$. Since the runtime is polynomial if ε is constant, the presented algorithm constitutes a PTAS w.r.t. the balance of the solution. As described in the introduction we introduce a coarse set of equivalence classes of the connected components, defined as follows.

Definition 4.1. Let $\mathcal{W}$ be a set of disjoint connected components of the vertices of T, and $\varepsilon > 0$. A vector $\vec{g} = (g_0, \ldots, g_t)$, where $t = \lceil \log_{1+\varepsilon}(1/\varepsilon) \rceil + 1$, is called the *signature* of $\mathcal{W}$ if in $\mathcal{W}$ there are g_0 components of size in the interval $[1, \varepsilon\lceil n/k \rceil[$ and g_i components of size in the interval $[(1+\varepsilon)^{i-1} \cdot \varepsilon\lceil n/k \rceil, (1+\varepsilon)^i \cdot \varepsilon\lceil n/k \rceil[$, for each $i \in \{1, \ldots, t\}$.

The first stage of our algorithm uses a dynamic programming scheme to find a set of connected components of minimum cut cost among those with signature $\vec{g}$, for any possible $\vec{g}$. Let $\mathfrak{W}$ denote the set containing each of these optimal sets that cover all vertices of the tree, as computed by the first stage. In the second stage the algorithm attempts to distribute the connected components in each set $\mathcal{W} \in \mathfrak{W}$ into k bins, where each bin has a capacity of $(1+\varepsilon)\lceil n/k \rceil$ vertices. This is done using a scheme originally proposed by Hochbaum and Shmoys [33, 69] for the `BIN PACKING` problem. The final output of our algorithm is the partition of the vertices of the given tree that corresponds to a packing of a set $\widetilde{\mathcal{W}} \in \mathfrak{W}$ that uses at most k bins and has minimum cut cost. Both stages of the algorithm have a runtime exponential in t. Hence the runtime is polynomial if ε is a constant.

4.2.1 The Cutting Phase

We now describe the dynamic programming scheme to compute the set of connected components of minimum cut cost among

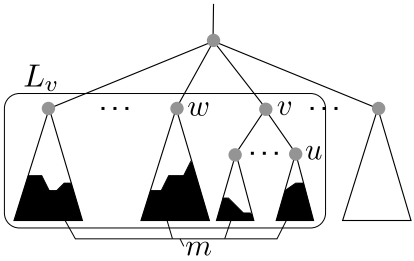


Figure 4.2: *A part of a tree in which a vertex v, its rightmost child u, its predecessor w among its siblings, the set of vertices L_v, and the m covered vertices by some lower frontier with signature $\vec{g}$ are indicated.*

those whose signature is $\vec{g}$, for every possible $\vec{g}$. We fix a root $r \in V$ among the vertices of T, and an ordering of the children of every vertex in V. We define the *leftmost* and the *rightmost* among the children of a vertex, the siblings *left of* a vertex, and the *predecessor* of a vertex among its siblings according to this order in the natural way. The idea is to recursively construct a set of disjoint connected components for every vertex $v \neq r$ by using the optimal solutions of the subtrees rooted at the children of v and the subtrees rooted at the siblings left of v. More formally, let for a vertex $v \neq r$ the set $L_v \subset V$ contain all the vertices of the subtrees rooted at those siblings of v that are left of v and at v itself (Figure 4.2). We refer to a set $\mathcal{F}$ of disjoint connected components as a *lower frontier of* L_v if all components in $\mathcal{F}$ are contained in L_v and the vertices in V not covered by $\mathcal{F}$ form a connected component containing the root r. For every vertex v and every signature $\vec{g}$, the algorithm recursively finds a lower frontier $\mathcal{F}$ of L_v with signature $\vec{g}$. Finally, a set of connected components with signature $\vec{g}$ covering all vertices of the tree can be computed using the solutions of the rightmost child of the root. The algorithm selects a set having minimum cut cost in each recursion step. Let $C_v(\vec{g}, m)$, for any vertex $v \neq r$ and any integer m, denote the optimal cut cost over those lower frontiers of L_v with signature $\vec{g}$ that cover a total of m vertices with their connected components. If no such set exists let $C_v(\vec{g}, m) = \infty$. Additionally, we define $\mu := (1+\varepsilon)\lceil n/k \rceil$, and

$\vec{e}(x)$ for any integer $x < \mu$ to be the signature of a set containing only one connected component of size x. We now show that the function $C_v(\vec{g}, m)$ can be computed using a dynamic program.

Let $\mathcal{F}^*$ denote an optimal lower frontier associated with $C_v(\vec{g}, m)$. We will consider the four cases resulting from whether or not the vertex v is a leaf, and whether or not it is the leftmost among its siblings. First consider the case when both properties are met. That is, v is a leaf and the leftmost among its siblings. Then $L_v = \{v\}$ and hence the set $\mathcal{F}^*$ either contains $\{v\}$ as a component or is empty. In the latter case the cut cost is 0. In the former it is $\omega(e)$ where e is the edge incident to the leaf that is cut from the tree. Thus $C_v((0, \ldots, 0), 0) = 0$ and $C_v(\vec{e}(1), 1) = \omega(e)$ while all other function values equal infinity. Now consider the case when v is neither a leaf nor the leftmost among its siblings. Let w be the predecessor among v's siblings and u the rightmost child of v. The set L_v contains the vertices of the subtrees rooted at v's siblings that are left of v and at v itself. The lower frontier $\mathcal{F}^*$ can either be one in which the edge from v to its parent is cut or not. In the latter case the m vertices that are covered by $\mathcal{F}^*$ do not contain v and hence are distributed among those in L_w and L_u since $L_v = L_w \cup L_u \cup \{v\}$. If x of the vertices in L_u are covered by $\mathcal{F}^*$ then $m - x$ must be covered by $\mathcal{F}^*$ in L_w. The vector $\vec{g}$ must be the sum of two signatures $\vec{g}_u$ and $\vec{g}_w$ such that the lower frontier of L_u (respectively L_w) has minimum cut cost among those having signature $\vec{g}_u$ (respectively $\vec{g}_w$) and covering x (respectively $m - x$) vertices. If this were not the case the lower frontier in L_u (respectively L_w) could be exchanged with an according one having a smaller cut cost—a contradiction to the optimality of $\mathcal{F}^*$. Hence in case v is a non-leftmost internal vertex and the edge to its parent is not cut,

$$C_v(\vec{g}, m) = \min \big\{ C_w(\vec{g}_w, m - x) + C_u(\vec{g}_u, x) \mid 0 \le x \le m \wedge \vec{g}_w + \vec{g}_u = \vec{g} \big\}. \quad (4.1)$$

If the edge connecting v to its parent is cut in $\mathcal{F}^*$, then all n_v vertices in the subtree rooted at v are covered by $\mathcal{F}^*$. Hence the other $m - n_v$ vertices covered by $\mathcal{F}^*$ must be included in L_w. Let x be the size of the component $W \in \mathcal{F}^*$ that includes v. Analogous to the case before, the lower frontiers L_u and L_w

with signatures $\vec{g}_u$ and $\vec{g}_w$ in $\mathcal{F}^* \setminus \{W\}$ must have minimum cut costs. Hence the vector $\vec{g}$ must be the sum of $\vec{g}_u$, $\vec{g}_w$, and $\vec{e}(x)$. Therefore in case the edge e to v's parent is cut,

$$C_v(\vec{g}, m) = \omega(e) + \min \big\{ C_w(\vec{g}_w, m - n_v) + C_u(\vec{g}_u, n_v - x) \mid 1 \leq x < \mu \wedge \vec{g}_w + \vec{g}_u + \vec{e}(x) = \vec{g} \big\}. \quad (4.2)$$

Taking the minimum value of the formulas given in (4.1) and (4.2) thus correctly computes the value for $C_v(\vec{g}, m)$ for the case in which v is neither a leaf nor the leftmost among its siblings. In the two remaining cases when v is either a leaf or a leftmost sibling, either the vertex u or w does not exist. For these cases the recursive definitions of $C_v(\cdot, \cdot)$ can easily be derived from Equations (4.1) and (4.2) by letting all function values $C_u(\vec{g}, x)$ and $C_w(\vec{g}, x)$ of a non-existent vertex u or w be 0 if $\vec{g} = (0, \ldots, 0)$ and $x = 0$, and ∞ otherwise.

The above recursive definitions for $C_v(\cdot, \cdot)$ give a framework for a dynamic programming scheme that computes the wanted solution set $\mathfrak{W}$ in polynomial-time if ε is a constant, as the next theorem shows.

Theorem 4.2. *For any tree T and any constant $\varepsilon > 0$ there is an algorithm that computes $\mathfrak{W}$ in polynomial time.*

Proof. If the tree contains only one vertex the theorem obviously holds. Otherwise the optimum solution from $\mathfrak{W}$ that has signature $\vec{g}$ must contain a connected component that includes the root r and has some size x. Clearly x is at least 1 and at most μ. Hence, if u denotes the rightmost child of the root r, the cut cost $C(\vec{g})$ of the optimal solution for $\vec{g}$ can be computed in linear time using the formula

$$C(\vec{g}) = \min\{C_u(\vec{g} - \vec{e}(x), n - x) \mid 1 \leq x < \mu\}. \quad (4.3)$$

An optimal set of connected components with signature $\vec{g}$ can be computed using the dynamic program given by the above equation together with the recursive definition of C_v by keeping track of the set of connected components used in each recursion step.

To analyse the runtime let us first bound the number of signatures $\vec{g}$ that have to be considered for a vertex v in the dynamic program. Let $N_v = |L_v|$ denote the number of vertices in the subtrees rooted at the siblings left of v and at v itself. There are N_v vertices that can be distributed into connected components of different sizes to form a lower frontier $\mathcal{W}$ of L_v. Each entry g_i of $\vec{g}$ counts components of size at least the lowest value of the i-th interval as specified in Definition 4.1. Hence each g_i is upper-bounded by $N_v/((1+\varepsilon)^{i-1} \cdot \varepsilon n/k) \leq k/((1+\varepsilon)^{i-1} \cdot \varepsilon)$ if $i \in \{1, \ldots, t\}$, and N_v if $i = 0$. Therefore the total number of signatures $\vec{g}$ considered for a vertex v is upper-bounded by

$$\begin{aligned} N_v \cdot \prod_{i=1}^{t} \frac{k}{(1+\varepsilon)^{i-1} \cdot \varepsilon} &= N_v \left(\frac{k}{\varepsilon}\right)^t \cdot \left(\frac{1}{1+\varepsilon}\right)^{\frac{(t-1)t}{2}} \\ &\leq N_v \left(\frac{k}{\sqrt{\varepsilon}}\right)^t, \end{aligned}$$

where the inequality holds since $t - 1 = \lceil \log_{1+\varepsilon}(1/\varepsilon) \rceil$. Because the latter value can be upper-bounded by $\lceil 1/\varepsilon \cdot \log(1/\varepsilon) \rceil$ if $\varepsilon \leq 1$, since then $1 + \varepsilon \geq 2^{\varepsilon}$, we can conclude that the number of signatures is σN_v, where $\sigma \in \mathcal{O}((k/\sqrt{\varepsilon})^{1+\lceil \frac{1}{\varepsilon} \log(\frac{1}{\varepsilon}) \rceil})$.

We bound the runtime as follows. For each vertex v we calculate the number of steps τ_v that are needed to compute all entries $C_{v'}(\vec{g}, m)$ for all $v' \in L_v$. We claim that $\tau_v \leq \frac{3}{2}\sigma^2 N_v^4$ for any vertex v. According to Formulas (4.1) and (4.2), in addition to the number of steps τ_u and τ_w to compute the tables for L_u and L_w, for each m and $\vec{g}$ the minimum value over two options is found by going through all possible x, $\vec{g}_u$, and $\vec{g}_w$. For any fixed x there are at most $\sigma N_u \cdot \sigma N_w$ many possibilities to combine vectors $\vec{g}_u$ and $\vec{g}_w$ to form a signature $\vec{g}$. Since m and x are both upper-bounded by N_v and $N_u + N_w \leq N_v$ we get

$$\begin{aligned} \tau_v &\leq \tau_u + \tau_w + 2\sigma^2 N_u N_w N_v^2 \\ &\leq \frac{3}{2}\sigma^2 N_v^2 \left(N_u^2 + N_w^2 + 2N_u N_w\right) \\ &\leq \frac{3}{2}\sigma^2 N_v^4. \end{aligned}$$

Since the time to compute Formula (4.3) for each signature is $\mathcal{O}(\sigma n)$, we conclude that the total runtime is $\mathcal{O}(\sigma^2 n^4) =$

$\mathcal{O}(n^4(k/\sqrt{\varepsilon})^{2+2\lceil\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon})\rceil})$, which is polynomial if ε is constant. □

4.2.2 The Packing Phase

The second stage of the algorithm attempts to pack each set of connected components $\mathcal{W} \in \mathfrak{W}$ computed by the first stage into k bins of capacity $(1+\varepsilon)\lceil n/k\rceil$. This means solving the well known `BIN PACKING` problem, which is NP-hard in general. However we are able to solve it in polynomial time for constant ε using a method developed by Hochbaum and Schmoys [33], which we briefly describe as presented in [69].

Let $\mathcal{W} \in \mathfrak{W}$ be a set of connected components with signature $\vec{g} = (g_0, \ldots, g_t)$. First the algorithm constructs an instance $\mathfrak{I}$ of the `BIN PACKING` problem containing only the components of $\mathcal{W}$ larger than $\varepsilon\lceil n/k\rceil$. In particular, the bin capacity is set to be $\lceil n/k\rceil$ and for every entry $1 \leq i \leq t$ of $\vec{g}$, g_i elements of size $(1+\varepsilon)^{i-1} \cdot \varepsilon\lceil n/k\rceil$ are introduced in $\mathfrak{I}$. That is, the size of each component is converted to the lower endpoint of the interval which contains it according to Definition 4.1. The number of elements in $\mathfrak{I}$ is $\sum_{i\geq 1} g_i \leq n/(\varepsilon\lceil n/k\rceil) \leq k/\varepsilon$ since there are n vertices in V and the smallest size of an element in $\mathfrak{I}$ is $\varepsilon\lceil n/k\rceil$. An optimal bin packing for $\mathfrak{I}$ can be found in $\mathcal{O}((k/\varepsilon)^{2t})$ time, using a dynamic programming scheme (for more details see [33, 69]). That is, the runtime is exponential in the number t of different sizes of the elements. A packing of $\mathfrak{I}$ into the minimum number of bins of capacity $\lceil n/k\rceil$ translates into a packing of the components of $\mathcal{W}$ larger than $\varepsilon\lceil n/k\rceil$ into bins of capacity $(1+\varepsilon)\lceil n/k\rceil$, since each element in $\mathfrak{I}$ underestimates the size of the component in $\mathcal{W}$ that it represents by a factor of at most $1+\varepsilon$.

To complete the packing of $\mathcal{W}$ the algorithm distributes the remaining components of size less than $\varepsilon\lceil n/k\rceil$ by greedily putting them into bins without exceeding the capacity of $(1+\varepsilon)\lceil n/k\rceil$. A new bin is created if placing a component in any of the bins would exceed the capacity. Distributing the remaining components can be performed in $\mathcal{O}(n)$ time. Let $\vartheta(\mathcal{W})$ denote

the number of bins that this algorithm needs to pack $\mathcal{W}$. Note that for two sets of components having the same signature the components larger than $\varepsilon\lceil n/k\rceil$ will always be distributed in the same way by the algorithm. However the greedy distribution of the remaining small components may create more bins for one of the sets. We show next that if a set of components computed by the first stage has the same signature $\vec{g}^*$ as the set of components induced by an optimal perfectly balanced partition, then the second stage of the algorithm packs it into at most k bins with capacity $(1+\varepsilon)\lceil n/k\rceil$.

Lemma 4.3. *Let $\mathcal{W}^*$ having signature $\vec{g}^*$ be the set of connected components in an optimal perfectly balanced partition. For the set $\mathcal{W} \in \mathfrak{W}$ with signature $\vec{g}^*$ it holds that $\vartheta(\mathcal{W}) \leq k$.*

Proof. We distinguish two cases for the greedy distribution of the components of $\mathcal{W}$ that have size less than $\varepsilon\lceil n/k\rceil$ depending on whether or not new bins are created. If no new bins are created then $\vartheta(\mathcal{W})$ is solely determined by the output of the bin packing algorithm, run with capacities $\lceil n/k\rceil$ on the instance $\mathfrak{I}$. Since $\mathcal{S}^*$ has the same signature $\vec{g}^*$ as $\mathcal{W}$, all elements $\mathfrak{e}_i \in \mathfrak{I}$ can be mapped to distinct components $W_i \in \mathcal{S}^*$ such that $\mathfrak{e}_i \leq |W_i|$. Hence any packing of $\mathcal{S}^*$ into bins of capacity $\lceil n/k\rceil$ requires at least $\vartheta(\mathcal{W})$ many bins which is optimal for $\mathfrak{I}$. Since $\mathcal{W}^*$ requires at most k optimally packed bins by definition, this proves the claim in the case no new bins are opened.

If new bins are created by the greedy step, then at least the first $\vartheta(\mathcal{W})-1$ bins of the final solution are filled beyond the extent of $\lceil n/k\rceil$. Otherwise small components of size at most $\varepsilon\lceil n/k\rceil$ could have been fit without requiring the creation of the last bin. Therefore the total number of vertices in $\mathcal{W}$ strictly exceeds $(\vartheta(\mathcal{W})-1)\lceil n/k\rceil$. Since the total number of vertices contained in $\mathcal{S}^*$ equals that of $\mathcal{W}$ it follows that at least $\vartheta(\mathcal{W})$ bins are required to pack $\mathcal{S}^*$ into bins of capacity $\lceil n/k\rceil$, which proves the claim in the case new bins were created by the greedy step. □

The final step of the algorithm is to output the packing of a set $\mathcal{W} \in \mathfrak{W}$ of minimum cut cost among those with $\vartheta(\mathcal{W}) \leq k$.

The next theorem proves correctness and bounds the runtime of the algorithm.

Theorem 4.4. *For any tree T with positive edge weights, $\varepsilon > 0$, and $k \in \{1, \ldots, n\}$, there is an algorithm that computes a partition of T's vertices into k sets such that each set has size at most $(1+\varepsilon)\lceil n/k \rceil$ and its cut cost is at most that of an optimal perfectly balanced partition of the tree. Furthermore the runtime is polynomial if ε is a constant.*

Proof. Let $\widetilde{\mathcal{W}} \in \mathfrak{W}$ be the set of connected components returned by the algorithm, i.e. if $C(\vec{g})$ denotes the cut cost of the set $\mathcal{W} \in \mathfrak{W}$ with signature $\vec{g}$, then

$$\widetilde{\mathcal{W}} = \arg_{\mathcal{W} \in \mathfrak{W}} \min\{C(\vec{g}) \mid \mathcal{W} \text{ has signature } \vec{g} \wedge \vartheta(\mathcal{W}) \leq k\}. \tag{4.4}$$

By Lemma 4.3 we know that if $\mathcal{W} \in \mathfrak{W}$ has signature $\vec{g}^*$ then $\vartheta(\mathcal{W}) \leq k$. Thus the minimisation of (4.4) ensures that the cut cost of $\widetilde{\mathcal{W}}$ is at most that of a set of components $\mathcal{W} \in \mathfrak{W}$ with signature $\vec{g}^*$. Since $\mathfrak{W}$ retains the set of components with minimum cut cost among all those having the same signature, it follows that the cut cost of $\widetilde{\mathcal{W}}$ is at most that of $\mathcal{W}^*$, which concludes the proof of correctness.

To bound the runtime of the second stage, recall from the proof of Theorem 4.2 that the total number of considered signatures $\vec{g}$ is σn, where $\sigma \in \mathcal{O}((k/\sqrt{\varepsilon})^{1+\lceil \frac{1}{\varepsilon} \log(\frac{1}{\varepsilon}) \rceil})$. By Theorem 4.2 the set $\mathfrak{W}$, whose size is at most σn, can be computed in time $\mathcal{O}(n^4\sigma^2)$. Each of the sets of components of $\mathfrak{W}$ requires at most $\mathcal{O}((k/\varepsilon)^{2t} + n)$ time to be packed in the second stage of the algorithm. Hence the second stage can be performed in $\mathcal{O}(\sigma n((k/\varepsilon)^{2t} + n))$ total time. This means that the overall runtime of the algorithm is $\mathcal{O}(n^4(k/\varepsilon)^{1+3\lceil \frac{1}{\varepsilon} \log(\frac{1}{\varepsilon}) \rceil})$, which concludes the proof. □

4.3 An Extension to Edge-Weighted Graphs

In this section we sketch an algorithm that employs the PTAS given in Section 4.2 to find near-balanced partitions for general graphs with positive edge weights. The cut cost computed is at most $\alpha \in \mathcal{O}(\log n)$ times that of an optimal perfectly balanced partition. The algorithm relies on using our PTAS to compute near-balanced partitions for a set of trees that well approximate the cuts in the given input graph G. These trees form a cut-based hierarchical decomposition of G and can be found using the results by Räcke [57]. The reason why this process generates an $\mathcal{O}(\log n)$ approximation of the cut cost depends on the properties of the decomposition, which we now sketch. A *decomposition tree* of an edge weighted graph G is an edge weighted tree T for which there is a one-to-one correspondence between the leaves of T and the vertices in G. Hence a partition of the vertices of G corresponds to a partition of the leaves of T, and vice versa. Accordingly we define a *leaf partition* $\mathcal{L}$ of a tree to be a partition of the leaf set. The *cut cost* $C(\mathcal{L})$ of a leaf partition $\mathcal{L}$ is the minimum weight of edges in T that have to be removed in order to disconnect the sets in $\mathcal{L}$ from each other. We make use of the following result which can be found in [48, 57]. It bounds the cut costs in a graph and its corresponding decomposition trees when cutting the graph into *two* parts.

Theorem 4.5 (follows from [48, 57])**.** *For any graph G with n vertices and positive edge weights, a family of decomposition trees $\{T_i\}_i$ of G and positive real numbers $\{\lambda_i\}_i$ such that $\sum_i \lambda_i = 1$ with the following properties can be found in polynomial time. Let an arbitrary partition of the vertices of G into two sets A and B be given. If $C(A, B)$ denotes its cut cost and $\mathcal{L}_i$ its corresponding leaf partition in T_i, we have*

- *the lower bound $C(\mathcal{L}_i) \geq C(A, B)$ for each i, and*
- *the upper bound $\sum_i \lambda_i C(\mathcal{L}_i) \leq \mathcal{O}(\log n) \cdot C(A, B)$.*

Since $\sum_i \lambda_i = 1$ the above theorem implies that for at least one tree T_i it holds that $C(\mathcal{L}_i) \leq \mathcal{O}(\log n) \cdot C(A, B)$. This

allows for the logarithmic approximation of the k-BALANCED PARTITIONING problem in graphs using a modified version of the PTAS given in Section 4.2. We adapt the PTAS to compute near-balanced leaf partitions of each T_i. That is, it computes a leaf partition of the l leaves of a tree T into k sets of size at most $(1+\varepsilon)\lceil l/k \rceil$ each. It is easy to see that the PTAS given in Section 4.2 can be adapted accordingly. First signatures need to count leaves instead of vertices in Definition 4.1. Moreover we need to keep track of the number l_v of leaves in a subtree of a vertex v instead of the number n_v of vertices in Equations (4.1) and (4.2). This yields the following result.

Corollary 4.6. *For any tree T with positive edge weights, $\varepsilon > 0$, and $k \in \{1, \dots, l\}$, there is an algorithm that computes a partition of the l leaves of T into k sets such that each set includes at most $(1+\varepsilon)\lceil l/k \rceil$ leaves and its cut cost is at most that of an optimal perfectly balanced leaf partition. Furthermore the runtime is polynomial in k and the number of vertices of T, if ε is constant.*

The algorithm computing near-balanced partitions for general edge-weighted graphs works as follows. According to Theorem 4.5 a family of decomposition trees can be computed for the given graph G in polynomial time. For each such tree we compute a near-balanced partition of its leaves into k sets using Corollary 4.6. We select the computed leaf partition having the smallest cut cost when applied to G. The output of the algorithm is then the corresponding vertex partition of G.

Using Theorem 4.5 it is possible to bound the weight C_j of an edge set cutting out a single part P_j of a partition $\mathcal{V}$. That is, we set A to part P_j and B to the union of all other parts in $\mathcal{V}$. Since the sum $\sum_{j=1}^{k} C_j$ is exactly twice the cut cost of the partition $\mathcal{V}$, Theorem 4.5 yields upper and lower bounds on the cut costs of $\mathcal{V}$ and its corresponding leaf partitions in the decomposition trees. Together with the bound on the cut cost of a computed leaf partition from Corollary 4.6, this fact makes it possible to prove the claimed approximation guarantee. A formal proof of the following theorem can be found in [24] and the doctoral thesis of Luca Foschini.

Theorem 4.7. *Let G be a graph with n vertices and positive edge weights, $\varepsilon > 0$ be a constant, and $k \in \{0, \ldots, n\}$. There is a polynomial time algorithm that computes a partition of the vertices of G into k sets such that each set has size at most $(1+\varepsilon)\lceil n/k \rceil$, and its cut cost is at most $\alpha \in \mathcal{O}(\log n)$ times that of the optimal perfectly balanced solution.*

4.4 Improving the Runtime and Other Observations

In this chapter, computing near-balanced solutions to the edge-weighted k-`BALANCED PARTITIONING` problem was studied on trees, and these results were applied to general graphs. Trees prove to be instances which admit a PTAS with approximation ratio $\alpha = 1$, the best possible in the bicriteria sense. Additionally the PTAS also enables us to show that an optimum perfectly balanced solution to the k-`BALANCED PARTITIONING` problem can be computed in polynomial time for trees when $k \in \Theta(n)$. This can be seen by for instance setting $\varepsilon = \lceil 1 + n/k \rceil^{-1}$ which is constant in this case. Hence the runtime of the PTAS is polynomial. It also means that the set sizes of the computed partitions will be smaller than $\lceil n/k \rceil + 1$, i.e. they contain at most $\lceil n/k \rceil$ vertices.

Using standard techniques [66], the dynamic program that cuts a tree into pieces in the first phase of the PTAS can also be adapted to bounded tree-width graphs. Hence also for this graph class a PTAS as the above one exists.

The ratio of $\alpha = 1$ also enables our PTAS for trees to be extended into an algorithm for general graphs with approximation factor $\alpha \in \mathcal{O}(\log n)$, improving on the best previous [1] bound of $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$. As noted in Section 3.1.1 (page 52) the same logarithmic approximation guarantee can be obtained for the `BISECTION` problem on general graphs. In fact this result was achieved in [57] by applying the algorithm by MacGregor [47], which computes optimal bisections for trees (see Section 2.1.2 page 14), to Theorem 4.5. Hence, remarkably, the same approxi-

mation guarantee can be obtained on the cut cost for the k-BALANCED PARTITIONING problem in case $k = 2$ (the BISECTION problem) and for unrestricted k, if we settle for near-balanced solutions in the latter case. This is in contrast to the strong inapproximability results [1] when the solutions are required to be perfectly balanced.

To improve the above approximation factors, the tree embedding results that we used unfortunately do not seem amenable. This is because the decomposition trees can also be used for the so called OBLIVIOUS ROUTING problem which is an online problem (cf. [57]). For it the gained competitive ratio of $\mathcal{O}(\log n)$ is optimal. This can be shown by a counter example [6, 49] in which any online strategy will have a logarithmic ratio. Furthermore, this counter example consists of a square shaped solid grid graph with corresponding predefined routes. This seems to suggest that the presented algorithm of this chapter does not perform better on solid grid graphs, even though it remains an open problem to prove this fact. In general, it seems as if radically different methods are needed to improve on the presented logarithmic approximation factor.

A remaining question is whether faster algorithms can be found to compute near-balanced partitions. We showed that we can achieve approximations on the cut cost that do not depend on ε. Hence it suggests itself to devise an algorithm that will compensate the cost of being able to compute a near-balanced solution for any $\varepsilon > 0$ not in the runtime but in the cut cost. The next chapter however shows that, unless P=NP, no such algorithm exists that is reasonable for practical applications.

Chapter 5

The Hardness of k-Balanced Partitioning

In this chapter we investigate the hardness of the k-BALANCED PARTITIONING[1] problem. We consider computing perfectly balanced and near-balanced partitions[2], and for both scenarios we give inapproximability results for a variety of graph classes. For the perfectly balanced case we show that, unless P=NP, neither for solid grid graphs[3] nor for trees polynomial time algorithms exist that approximate the cut size within any factor that is reasonable for practical applications. For trees we extend this result by considering constant degrees. In particular we show that the problem is NP-hard for trees with maximum degree 5, and that it is APX-hard when the degree is upper-bounded by 7. Additionally we consider computing near-balanced partitions for any given $\varepsilon > 0$ on trees, solid grids, and general graphs. We concern ourselves with *fully polynomial time* algorithms for which the cut size is approximated within a factor that may

[1] Definition 1.1 page 3
[2] Definition 3.1 page 51
[3] Definition 1.2 page 4

increase when the bound on the set sizes becomes more stringent. We will show that, unless P=NP, for the considered graph classes no such algorithm exists that is reasonable for practical applications. We develop a reduction framework which implies all but the results for constant degree trees. It identifies some necessary conditions on the considered graphs which make them hard for k-BALANCED PARTITIONING.

The results in the first part of this chapter (Sections 5.2 and 5.3) were published in [21] and are the sole work of the author of this dissertation. The second part (Section 5.4) was published as an extended abstract [24] and is joint work with Luca Foschini. Some of the results in the second part will partially appear in the doctoral thesis of Luca Foschini. In particular the proof for trees of degree 5 (Theorem 5.13) will only be sketched here. The details can be found in [24] and the thesis of Luca Foschini.

5.1 The End of the Road

In this thesis we have so far seen algorithms that solve the k-BALANCED PARTITIONING problem in a variety of settings. In particular for arbitrary values of k, in Chapter 3 we presented an algorithm that is fast on solid grid graphs and approximates the cut size within a logarithmic factor. However the computed partition can deviate by a factor of 2 from being perfectly balanced. For this result we harnessed some known techniques which were developed for general graphs. The methods were originally devised since, by a result of Andreev and Räcke [1], in general it is NP-hard to approximate the cut size within any finite factor if perfectly balanced solutions are desired. However this hardness result does not exclude the existence of approximation algorithms that compute partitions which are closer to being perfectly balanced than a factor of 2 as achieved in Chapter 3. We therefore concerned ourselves with near-balanced solutions for arbitrary constants $\varepsilon > 0$ in Chapter 4. We gave an algorithm that for general graphs approximates the cut size within a logarithmic factor. Even though it runs in polynomial

time, it is slow since the runtime increases exponentially with decreasing ε.

The aforementioned inapproximability result by Andreev and Räcke [1] for perfectly balanced solutions relies on the fact that a general graph may be disconnected. Hence their reduction is not feasible when considering connected graph classes such as solid grid graphs or trees. In this chapter we will show however, that approximating the cut size of the k-BALANCED PARTITIONING problem remains similarly hard for these graph classes. In particular we prove that for arbitrary k, computing perfectly balanced partitions for solid grid graphs is NP-hard even when approximating the cut size within a factor of $\alpha = n^c$, for an arbitrary constant $c < 1/2$. For trees we show that this is true even if $c < 1$. As we will see, both of these results are asymptotically tight.

The latter inapproximability result for trees is even valid if the *diameter*, i.e. the maximum distance between two leaves in terms of edges, is bounded by 4. It is easy to see that the k-BALANCED PARTITIONING problem is trivial on trees of diameter at most 3. Hence also in this respect our result is tight. However the proof relies on the fact that the trees used in the reduction can have arbitrary degrees. It is therefore an interesting question whether the k-BALANCED PARTITIONING problem remains hard when the maximum degree Δ of the trees is bounded. Obviously for trees with $\Delta = 2$, i.e. paths, the problem is trivial. However we are able to show that the problem is NP-hard if $\Delta = 5$, and that it is APX-hard when $\Delta = 7$. These observations are in contrast with an algorithm that will compute perfectly balanced partitions for trees while approximating the cut size within a factor of $\alpha \in \mathcal{O}(\Delta \log_\Delta(n/k))$. This algorithm can be derived from the work of MacGregor [47] and means that for constant degree trees a logarithmic approximation can be computed.

An additional observation when considering the algorithms for solid grid graphs presented in this thesis so far, is the following. On one hand we can solve the k-BALANCED PARTITIONING problem fast if we allow the partition to be off by a factor of 2 from being perfectly balanced. On the other hand we can compute near-balanced solutions for arbitrary constants $\varepsilon > 0$ when

we allow considerably slower algorithms in which the exponent of the runtime depends on ε. Ideally we would want an algorithm that computes near-balanced solutions in *fully polynomial time* such that the runtime does not increase exponentially when ε decreases. More precisely, we want the runtime to be at most $\pi(n/\varepsilon)$ for some polynomial $\pi(\cdot)$.

One interesting property of the algorithm computing near-balanced solutions is that its approximation factor $\alpha \in \mathcal{O}(\log n)$ on the cut size does not depend on ε. It therefore suggests itself to devise an algorithm that will compensate the cost of being able to compute a near-balanced solution for any $\varepsilon > 0$ not in the runtime but in the cut size. In the following however, we show that no reasonable algorithm of this sort exists. More precisely, we show that for solid grid graphs there is no fully polynomial time algorithm with the following properties, unless P=NP. The solution computed by the supposed algorithm should be near-balanced for any given $\varepsilon > 0$, and the cut size may deviate from the optimum of a perfectly balanced partition by a factor of $\alpha = n^c/\varepsilon^d$, for any constants c and d where $c < 1/2$. We will also show that for trees an analogous statement is true even if $c < 1$. For general graphs (which may be disconnected) we can prove that no such algorithm exists for any finite α. To the best of our knowledge, these are the first bicriteria inapproximability results for the k-`BALANCED PARTITIONING` problem. We will show that all of these results are asymptotically tight by providing corresponding approximation algorithms.

5.1.1 An Overview of the Used Techniques

In the first part of this chapter we will give all the above mentioned hardness results, except those considering constant degree trees. The main contribution of this part is a reduction framework with which these hardness results can be generated. In particular, we identify some sufficient conditions that a graph class has to fulfil in order to be hard for k-`BALANCED PARTITIONING`. Intuitively these conditions say that it is expensive to cut out vertices in terms of the used edges. We will show that graphs fulfilling the conditions can be used in order to decide

the 3-PARTITION problem. In this problem a set of integers and a threshold parameter are given. The integers need to be partitioned into triples such that each triple sums up to exactly the threshold value. This problem is known to be strongly NP-hard [30]. We will assume that we can construct a graph from each 3-PARTITION instance such that the conditions are fulfilled. We then state a lemma which asserts that an algorithm which computes near-balanced partitions and approximates the cut size within some α on the constructed graphs, is able to decide whether all integers of a 3-PARTITION instance can be partitioned into the desired triples. By parametrising the involved bounds of the conditions on the respective approximation factors and showing that for these values the graphs can be constructed in polynomial time, we yield the above mentioned inapproximability results.

For grids and trees the sufficient conditions of the reduction framework are met as follows. As also elaborated on in Chapters 2 and 3.2, a grid graph resembles a discretised polygon. They therefore also share similar isoperimetric properties. We are able to use these in order to show that cutting grid graphs with few edges can only cut off a limited number of vertices. For trees we instead rely on an entirely different property, namely their ability to have high vertex degrees. Since from a combinatorial point of view these graphs are simple but also very different, this demonstrates the capability of the reduction framework to cover a wide spectrum of graph classes.

Since for trees the framework relies on high vertex degrees, we need to employ an entirely different reduction technique in the second part of this chapter to show our results for constant degree trees. In particular the APX-hardness proof for trees of degree 7 uses the version of 3-PARTITION that has a constant sized gap. That is, it needs to be decided whether all or at most a constant fraction of the integers can be partitioned into triples that sum up to the threshold value. This problem is called GAP-3-PARTITION and is NP-hard [30, 56]. We will use a gap-preserving reduction for our result. For this we must rely on the structure of the constructed gadgets in order to make the respective reductions work. In particular we need gadgets that

guarantee a number of cut edges proportional to the number of integers that cannot be packed into triples in a GAP-3-PARTITION instance. For trees of maximum degree 5 we use similar ideas in a reduction from 3-PARTITION to show that k-BALANCED PARTITIONING is NP-hard.

5.1.2 Related Work

For a thorough review of related results regarding the k-BALANCED PARTITIONING problem see Section 4.1.2. This chapter significantly extends the known hardness results for the problem. Garey *et al.* [31] showed that the problem is NP-hard even for the restricted case when $k = 2$, i.e. the BISECTION problem. Additionally Andreev and Räcke [1] proved that for general k it is NP-hard to approximate the optimum cut size on general graphs when perfectly balanced solutions are required. This is done by a reduction from the 3-PARTITION problem. For each of the k integers of a 3-PARTITION instance a connected graph of size exactly the value of the integer is constructed. The disconnected graph containing these k graphs then constitutes the reduced instance. Since 3-PARTITION is strongly NP-hard [30] the graph can be computed in polynomial time. It is then easy to see that the optimum perfectly balanced solution does not cut any edge if the 3-PARTITION instance is solvable. Otherwise at least one edge must be cut and hence no polynomial time approximation algorithm can compute a solution, unless P=NP.

For trees of maximum degree Δ it is known that a perfectly balanced partition that approximates the cut size within $\alpha \in \mathcal{O}(\Delta \log_\Delta(n/k))$ can be computed in polynomial time. This follows from the results of MacGregor [47]. He devises a greedy algorithm that will cut any desired number of vertices from a tree recursively. It is well-known that every tree has a vertex which, when removed from the tree, leaves only subtrees that each contain at most half the vertices of the tree [45]. In a recursion step the algorithm by MacGregor will find this vertex in linear time and sort the corresponding subtrees by size. It will then cut off the subtrees in decreasing order until too many vertices were taken. To correct for the vertices that are superfluous the

algorithm will continue recursively on the last cut off subtree. In each recursion step at most Δ edges are cut, while the recursion depth is logarithmic. This is because the subtree on which the recursion continues is always at most half the size of the current tree. Using a slightly more sophisticated analysis of the recursion depth one can conclude that at most $\mathcal{O}(\Delta \log_\Delta(n/k))$ edges are cut. Now this recursive procedure is used k times for each part of the desired partition and hence the total cut size is $\mathcal{O}(k\Delta \log_\Delta(n/k))$. Any perfectly balanced solution has at least $\frac{n}{\lceil n/k \rceil} > k/2$ parts. Since the number of parts is integer and the tree is connected, this means at least $k/2$ edges must be cut by the optimum. Hence the approximation ratio of $\alpha \in \mathcal{O}(\Delta \log_\Delta(n/k))$ follows. MacGregor originally devised this algorithm to show an upper bound on the bisection width of trees. He also shows that there are perfect ternary trees that asymptotically match the logarithmic upper bound [47, 53].

5.2 A Reduction Framework

To derive the hardness results we give a reduction from the `3-PARTITION` problem defined below. It is known that this problem is strongly NP-hard [30] which means that it remains so even if all integers are polynomially bounded in the size of the input.

Definition 5.1 (`3-PARTITION`)**.** Given $3k$ integers a_1 to a_{3k} and a threshold s such that $s/4 < a_i < s/2$ for each $i \in \{1, \ldots, 3k\}$, and $\sum_{i=1}^{3k} a_i = ks$, find a partition of the integers into k triples such that each triple sums up to exactly s.

We will set up a general framework for a reduction from `3-PARTITION` to different graph classes. This will be achieved by identifying some structural properties that a graph constructed from a `3-PARTITION` instance has to fulfil, in order to show the hardness of the k-`BALANCED PARTITIONING` problem. While describing the structural properties we will exemplify them for (disconnected) general graphs which constitute an easily understandable case. For these graphs Andreev and Räcke [1] already

showed that it is NP-hard to approximate the cut size within any finite factor. We will show that, unless P=NP, no fully polynomial time algorithm exists even if the balance is approximated as well.

For any 3-PARTITION instance we construct a set of $3k$ graphs, which we will call *gadgets*, with a number of vertices proportional to the integers a_1 to a_{3k}. In particular, for general graphs each gadget H_i, where $i \in \{1, \dots 3k\}$, is a connected graph on $2a_i$ vertices. This assures that the gadgets can be constructed in polynomial time since 3-PARTITION is strongly NP-hard. In general we will assume that we can construct $3k$ gadgets for the given graph class such that each gadget has μa_i vertices for some μ specific for the graph class. These gadgets will then be connected using some number g of edges. The parameters μ and g may depend on the values of the given 3-PARTITION instance. For the case of general graphs we chose $\mu = 2$ and we let $g = 0$, i.e. the gadgets are disconnected. In order to show that the given gadgets can be used in a reduction, we will require that an upper bound can be given on the number of vertices that can be cut out using a limited number of edges. More precisely, given any colouring of the vertices of all gadgets into k colours, by a *minority vertex* in a gadget H_i we mean a vertex that has the same colour as less than half of H_i's vertices. Any partition of the vertices of all gadgets into k sets induces a colouring of the vertices into k colours. For approximation ratios α and ε, the property we need is that cutting the graph containing n vertices into k parts using at most αg edges, produces less than $\mu - \varepsilon n$ minority vertices in total. Clearly ε needs to be sufficiently small so that the graph exists. When considering fully polynomial time algorithms, ε should however also not be too small since otherwise the runtime may not be polynomial. For general graphs we achieve this by choosing $\varepsilon = (2ks)^{-1}$. This means that $\mu - \varepsilon n = 1$ since $n = \sum_{i=1}^{3k} \mu a_i = 2ks$. Simultaneously the runtime of a corresponding algorithm is polynomial in the size of the 3-PARTITION instance since 3-PARTITION is strongly NP-hard. Additionally the desired condition is met for this graph class since no gadget can be cut using $\alpha g = 0$ edges. The following definition formalises the properties needed for our reductions.

Definition 5.2 (reduction set)**.** For each instance $\mathfrak{I}$ of 3-PARTITION with integers a_1 to a_{3k} and threshold s, a *reduction set for k-BALANCED PARTITIONING* contains a graph determined by some given parameters g, μ, ε, and α which may depend on $\mathfrak{I}$. Such a graph constitutes $3k$ gadgets connected through g edges. Each gadget H_i, where $i \in \{1, \ldots, 3k\}$, has μa_i vertices. Additionally, if a partition of the n vertices of the graph into k sets has a cut size of at most αg, then in total there are less than $\mu - \varepsilon n$ minority vertices in the induced colouring.

Obviously the involved parameters have to be set to appropriate values in order for the reduction set to exist. For instance μ must be an integer and ε must be sufficiently small compared to μ and n. Since however the values will vary with the considered graph class we fix them only later. In the following lemma we will assume that the reduction set exists and therefore all parameters were chosen appropriately. It assures that given a reduction set, an approximation algorithm for k-BALANCED PARTITIONING can decide the 3-PARTITION problem. For general graphs we have seen above that a reduction set exists for any finite α and $\varepsilon = (2ks)^{-1}$. This means that a fully polynomial time algorithm for k-BALANCED PARTITIONING computing near-balanced partitions and approximating the cut size within α, can decide the 3-PARTITION problem in polynomial time. Such an algorithm can however not exist, unless P=NP.

Lemma 5.3. *Let for $\varepsilon \geq 0$ an algorithm* **A** *be given that for any graph in a reduction set for k-BALANCED PARTITIONING computes a partition of the n vertices into k parts of size at most $(1+\varepsilon)\lceil n/k \rceil$ each. If the cut size of the computed solution deviates by at most α from the optimal cut size of a perfectly balanced solution, then the algorithm can decide the 3-PARTITION problem.*

Proof. Let k be the value given by a 3-PARTITION instance $\mathfrak{I}$, and let G be the graph corresponding to $\mathfrak{I}$ in the reduction set. Assume that $\mathfrak{I}$ has a solution. Then obviously cutting the g edges connecting the gadgets of G gives a perfectly balanced solution to $\mathfrak{I}$. Hence in this case the optimal solution has a cut size of at most g. Accordingly algorithm **A** will cut at most αg

edges since it approximates the cut size by a factor of α. We will show that in the other case when $\mathfrak{I}$ does not have a solution, the algorithm will cut more than αg edges. Hence **A** can decide the 3-PARTITION problem and thus the lemma follows.

For the sake of deriving a contradiction assume that algorithm **A** cuts at most αg edges in case the 3-PARTITION instance $\mathfrak{I}$ does not permit a solution. Since the corresponding graph G is from a reduction set for k-BALANCED PARTITIONING, by Definition 5.2 this means that from its n vertices, in total less than $\mu - \varepsilon n$ are minority vertices in the colouring induced by the computed solution of **A**. Each gadget H_i, where $i \in \{1, \dots 3k\}$, of G has a *majority colour*, i.e. a colour that more than half the vertices in H_i share. This is because the size of H_i is μa_i and we can safely assume that $a_i \geq 2$ (otherwise the 3-PARTITION instance is trivial due to $s/4 < a_i < s/2$). The majority colours of the gadgets induce a partition $\mathcal{I}$ of the integers a_i of $\mathfrak{I}$ into k sets. That is, we introduce a set in $\mathcal{I}$ for each colour and put an integer a_i in a set if the majority colour of H_i equals the colour of the set.

Since we assume that $\mathfrak{I}$ does not admit a solution, if every set in $\mathcal{I}$ contains exactly three integers there must be some set for which the contained integers do not sum up to exactly the threshold s. On the other hand the bounds on the integers, assuring that $s/4 < a_i < s/2$ for each $i \in \{1, \dots, 3k\}$, mean that in case not every set in $\mathcal{I}$ contains exactly three elements, there must also exist a set for which the contained numbers do not sum up to s. By the pigeonhole principle and the fact that the sum over all a_i equals ks, there must thus be some set I among the k in $\mathcal{I}$ for which the sum of the integers is strictly less than s. Since the involved numbers are integers we can conclude that the sum of the integers in I is in fact at most $s - 1$. Therefore the number of vertices in the gadgets corresponding to the integers in I is at most $\mu(s - 1)$. Let w.l.o.g. the colour of I be 1. Apart from the vertices in these gadgets having majority colour 1, all vertices in G that also have colour 1 must be minority vertices. Hence there must be less than $\mu(s - 1) + \mu - \varepsilon n$ many vertices with colour 1. Since $\sum_{i=1}^{3k} a_i = ks$ and thus $\mu s = n/k$, these are less than $n/k - \varepsilon n$.

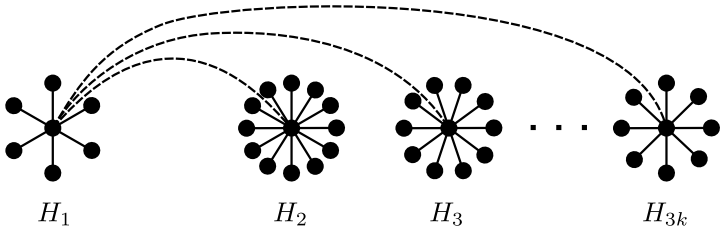


Figure 5.1: *The tree constructed for the reduction from* 3-PARTITION. *Each gadget H_i is a star of a_i vertices. All centre vertices are connected through H_1 which means that the resulting tree has diameter* 4.

At the same time the algorithm computes a solution inducing a colouring in which each colour has at most $(1+\varepsilon)n/k$ vertices, since $n = \mu ks$ is divisible by k. This means we can give a lower bound of $n-(k-1)(1+\varepsilon)n/k$ on the number of vertices of a colour by assuming that all other colours have the maximum number of vertices. Since this lower bound equals $(1+\varepsilon)n/k-\varepsilon n$, for any $\varepsilon \geq 0$ we get a contradiction on the upper bound derived above for colour 1. Thus the assumption that the algorithm cuts less than αg edges if $\mathfrak{I}$ does not have a solution is wrong. □

5.3 The Hardness for Grids and Trees

We will now consider some specific graph classes and show that the k-BALANCED PARTITIONING problem is hard when restricted to them. In particular we will first consider trees and thereafter solid grid graphs. For both graph classes we will first identify the reduction set for k-BALANCED PARTITIONING and then prove the respective hardness result. The following lemma gives the reduction set for trees, in which each graph is a star of stars as shown in Figure 5.1. These have high degrees and hence cutting off vertices is expensive in terms of the amount of used edges.

Lemma 5.4. *Let $\varepsilon \geq 0$ and $\alpha \geq 1$. For any* 3-PARTITION *instance with integers a_1 to a_{3k} and threshold s, construct a*

tree that consists of $3k$ *stars such that the number of vertices in each star* H_i *is* μa_i*, where* $i \in \{1, \ldots, 3k\}$ *and* $\mu = \lceil 3k\alpha + \varepsilon n \rceil$*. Moreover each centre vertex of a star* H_i*, for* $i \geq 2$*, is connected to the centre vertex of* H_1 *through an edge. If they exist, these trees form a reduction set for* k*-BALANCED PARTITIONING.*

Proof. Note that the gadgets of each tree are connected by $g = 3k-1$ edges. Using at most αg, i.e. less than $3k\alpha$, edges to cut off vertices from a single star H_i will cut off less than $3k\alpha$ leaves. We can safely assume that $a_i \geq 2$ for each $i \in \{1, \ldots, 3k\}$ since by $s/4 < a_i < s/2$ the 3-PARTITION instance would otherwise be trivial. Hence more than half the vertices of the star is still connected to the centre vertex since each star contains at least $6k\alpha$ vertices. Therefore a partition of the vertices of all gadgets into k sets with cut size at most αg will in total produce less than $3k\alpha$ minority vertices in the induced colouring. This establishes the desired upper bound on the number of minority vertices for the reduction set since $3k\alpha \leq \mu - \varepsilon n$. □

Using the above reduction set, in the following theorem[4] we show that it is NP-hard to approximate the cut size on trees within any satisfying factor. Note that the trees in the reduction set have diameter 4.

Theorem 5.5. *There is no polynomial time algorithm for the* k*-BALANCED PARTITIONING problem on trees that approximates the cut size within* $\alpha = n^c$ *for any constant* $c < 1$*, unless P=NP. This is true even if the diameter of the tree is at most* 4*.*

Proof. We need to show that a reduction set as proposed in Lemma 5.4 exists in order to use it together with Lemma 5.3. To prove the existence we show that the number of vertices of a tree as suggested by Lemma 5.4 is finite and hence its construction is feasible. Since the balance of the solution is not to be approximated we set $\varepsilon = 0$ which means that $\mu = \lceil 3k\alpha \rceil$. We assume w.lo.g. that $\alpha \geq 1$ and hence $\mu \leq 6k\alpha$. By letting

[4] An alternative proof to this theorem appears in [24] and the doctoral thesis of Luca Foschini.

$\alpha = n^c$ the number of vertices is

$$n = \sum_{i=1}^{3k} \mu a_i \leq 6k^2 s n^c.$$

Solving this inequality for n gives $n \leq (6k^2 s)^{\frac{1}{1-c}}$ which is finite since $c < 1$. Additionally n is polynomial in the size of the `3-PARTITION` instance since c is a constant and `3-PARTITION` is strongly NP-hard. By Lemma 5.3 a polynomial time algorithm that computes a perfectly balanced partition on any tree given by the reduction set and approximates the cut size within α can decide the `3-PARTITION` problem. Since the trees can be constructed in polynomial time this gives a contradiction unless P=NP, which concludes the proof. □

Next we show that no satisfactory fully polynomial time approximation algorithm exists that computes near-balanced solutions.

Theorem 5.6. *For the k-`BALANCED PARTITIONING` problem on trees, unless P=NP there is no fully polynomial time algorithm that for any $\varepsilon > 0$ computes a solution in which each of the k sets has size at most $(1+\varepsilon)\lceil n/k \rceil$ and for which $\alpha = n^c/\varepsilon^d$, for any constants c and d where $c < 1$. This is true even if the diameter of the tree is at most 4.*

Proof. In order to prove the claim we need to show that a reduction set as suggested by Lemma 5.4 exists and can be constructed in polynomial time. To prove the existence we show that the number of vertices of a tree as suggested by Lemma 5.4 is finite and hence its construction is feasible. We assume w.l.o.g. that $\alpha \geq 1$ and hence $\mu = \lceil 3k\alpha + \varepsilon n \rceil \leq 6k\alpha + 2\varepsilon n$. Since the algorithm can compute a near-balanced partition for any $\varepsilon > 0$ we set $\varepsilon = (4ks)^{-1}$. Thus the number of vertices in a tree of the reduction set is

$$n = \sum_{i=1}^{3k} \mu a_i \leq 6k^2 s\alpha + 2ks\varepsilon n = 6k^2 s(4ks)^d n^c + n/2.$$

Solving this inequality for n gives $n \leq (12k^2 s(4ks)^d)^{\frac{1}{1-c}}$ which is finite since $c < 1$. Additionally if c and d are constants, any

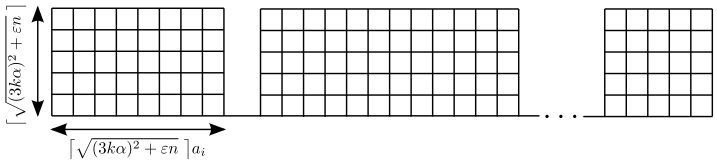


Figure 5.2: *The solid grid graph constructed for the reduction from* 3-PARTITION*. The gadgets which are rectangular grids are connected through the bottom left and right vertices.*

tree in the reduction set can be constructed in polynomial time since 3-PARTITION is strongly NP-hard. For $\varepsilon = (4ks)^{-1}$ a fully polynomial time algorithm has a runtime that is polynomial in the 3-PARTITION instance when executed on the corresponding tree in the reduction set. However, unless P=NP, this algorithm cannot exist since it decides the 3-PARTITION problem due to Lemma 5.3. □

Lemma 5.4 shows that high degree vertices can lead to inapproximability results for the k-BALANCED PARTITIONING problem. Grid graphs do not have high degrees. Instead however, we are able to harness their isoperimetric properties to gain similar hardness results. We establish these by using a set of *rectangular* grid graphs which are connected in a row (Figure 5.2). By a *rectangular* grid graph we mean a solid grid graph with the following properties. In its natural planar embedding for which the vertices are coordinates in $\mathbb{N}^2$ and the edges have unit length, the straight line edges touching the exterior face together form an orthogonal rectangle. The *width* of a rectangular grid graph is the number of vertices sharing the same y-coordinate in this embedding. Accordingly the *height* is the number sharing the same x-coordinate. We first prove that such topologies can be used as reduction sets for k-BALANCED PARTITIONING.

Lemma 5.7. *Let $\varepsilon \geq 0$ and $\alpha \geq 1$. For any* 3-PARTITION *instance, let a solid grid graph be given that consists of $3k$ rectangular grids which are connected in a row using their lower left and lower right vertices. Moreover let the height and width of a rectangular grid H_i, where $i \in \{1, \dots 3k\}$, be $\lceil\sqrt{(3k\alpha)^2+\varepsilon n}\rceil$*

and $\lceil \sqrt{(3k\alpha)^2 + \varepsilon n} \rceil a_i$*, respectively. If they exist, these grid graphs form a reduction set for* k*-*`BALANCED PARTITIONING`*.*

Proof. Consider one of the described graphs G for a `3-PARTITION` instance, and notice that its gadgets are connected using $g = 3k - 1$ edges. Since both the height and the width of each rectangular grid H_i is greater than αg, using at most αg edges it is not possible to cut across a gadget H_i, neither in horizontal nor in vertical direction. Consider the polygon $\mathscr{P}_{H_i}$ that can be constructed from H_i as described in Chapter 3 (cf. Figure 3.3). Under the condition that the cut size must be smaller than the height and the width of the rectangular polygon $\mathscr{P}_{H_i}$, the maximum area can be cut out from $\mathscr{P}_{H_i}$ by a square shaped cut in one of its corners. This is due to the isoperimetric properties [67] of $\mathscr{P}_{H_i}$. The size of the resulting cut out area is an upper bound on the number of vertices that can be cut out from H_i with the same cut size. This also follows from [54, Lemma 2] in which the maximum number of vertices that can be cut out from a grid graph using a fixed number of edges is determined. Hence using at most αg edges, at most $(\alpha g/2)^2$ vertices can be cut out from H_i. This also means that in any colouring induced by a partition of the vertices of the grid graph G into k sets with cut size at most αg, there are at most $(\alpha g/2)^2$ minority vertices in total. Since the size of each gadget is its height times its width, the parameter μ is greater than $(\alpha g)^2 + \varepsilon n$. Hence the number of minority vertices is less than $\mu - \varepsilon n$. □

The above topology is used with varying sizes of the gadgets in order to prove the desired results. We first show that it is NP-hard to approximate the cut size within any satisfying factor.

Theorem 5.8. *There is no polynomial time algorithm for the* k*-*`BALANCED PARTITIONING` *problem on solid grid graphs that approximates the cut size within* $\alpha = n^c$ *for any constant* $c < 1/2$*, unless P=NP.*

Proof. We need to show that a reduction set as proposed in Lemma 5.7 exists in order to use it together with Lemma 5.3. To prove the existence we show that the number of vertices

of a grid as suggested by Lemma 5.7 is finite and hence its construction is feasible. Since the balance of the solution is not to be approximated we set $\varepsilon = 0$. The parameter μ is determined by the height and width of the gadgets which in this case are $\lceil 3k\alpha \rceil$ and $\lceil 3k\alpha \rceil a_i$, respectively, for a gadget H_i. We assume w.l.o.g. that $\alpha \geq 1$ which gives $\mu = \lceil 3k\alpha \rceil^2 \leq (6k\alpha)^2$. The number of vertices of the resulting grid is

$$n = \sum_{i=1}^{3k} \mu a_i \leq (6kn^c)^2 ks.$$

Solving this inequality for n gives $n \leq (36k^3 s)^{\frac{1}{1-2c}}$ which is finite if $c < 1/2$. Additionally it is polynomial in the size of the `3-PARTITION` instance since c is a constant and `3-PARTITION` is strongly NP-hard. By Lemma 5.3 a polynomial time algorithm that computes a perfectly balanced partition on any grid given by the reduction set and approximates the cut size within α can decide the `3-PARTITION` problem. Since the grids can be constructed in polynomial time this gives a contradiction unless P=NP, which concludes the proof. □

Finally we show that no fully polynomial time algorithm exists that approximates the optimum cut size within any reasonable factor for solid grid graphs.

Theorem 5.9. *For the k-`BALANCED PARTITIONING` problem on solid grid graphs, unless P=NP there is no fully polynomial time algorithm that for any $\varepsilon > 0$ computes a solution in which each of the k sets has size at most $(1+\varepsilon)\lceil n/k \rceil$ and for which $\alpha = n^c/\varepsilon^d$, for any constants c and d where $c < 1/2$.*

Proof. Again, we prove the existence of the reduction set given in Lemma 5.7 by showing that the number of vertices of a grid suggested by the lemma is finite. Assuming w.l.o.g. that $\alpha \geq 1$ we get $\left\lceil \sqrt{(3k\alpha)^2 + \varepsilon n} \right\rceil \leq 2\sqrt{(3k\alpha)^2 + \varepsilon n}$. Hence the parameter μ, which is determined by the width and height of the gadgets, is at most $4(3k\alpha)^2 + 4\varepsilon n$. We assume that a fully polynomial algorithm exists which computes near-balanced partitions for any $\varepsilon > 0$. Therefore we may set $\varepsilon = (8ks)^{-1}$ which also means

that the runtime of the algorithm is polynomial in the size of the 3-PARTITION instance, since 3-PARTITION is strongly NP-hard. The number of vertices can be upper-bounded by

$$\begin{aligned} n &= \sum_{i=1}^{3k} \mu a_i \\ &\leq (4(3kn^c/\varepsilon^d)^2 + 4\varepsilon n)ks \\ &= 36k^3 s(8ks)^{2d} n^{2c} + n/2. \end{aligned}$$

Solving this inequality for n gives $n \leq (72k^3 s(8ks)^{2d})^{\frac{1}{1-2c}}$ which is finite since $c < 1/2$. Also it is polynomial if c and d are constants since 3-PARTITION is strongly NP-hard. Therefore the algorithm can decide the 3-PARTITION problem in polynomial time due to Lemma 5.3. This however is a contradiction unless P=NP, which concludes the proof. □

5.4 The Hardness for Constant Degree Trees

We now consider the problem of finding a perfectly balanced partition of a tree when the tree has constant degree. We will show hardness results for this problem when the maximum degree is 5 and when it is 7. The first reduction for maximum degree 7 is from the GAP-3-PARTITION problem. This is the 3-PARTITION problem in which, for a given ρ, either all or at most a ρ fraction of the integers can be partitioned into the desired triples. Formally it is defined as follows.

Definition 5.10 (GAP-3-PARTITION)**.** Let $3k$ integers a_1 to a_{3k}, a threshold s, and $\rho > 1$ be given, such that $s/4 < a_i < s/2$, $\sum_{i=1}^{3k} a_i = ks$, and the integers can either be partitioned into k triples that sum up to exactly s or at most k/ρ of them. Decide whether k or at most k/ρ such triples can be found.

There is a constant $\rho > 1$ such that the GAP-3-PARTITION problem is NP-hard. This is true even if all integers are polynomially bounded in k. These results follow from the original

NP-hardness proof of 3-PARTITION by Garey and Johnson [30] and the results by Petrank [56]. The latter result introduces a constant sized gap for the 3D-MATCHING problem. By considering the reductions given by Garey and Johnson from 3D-MATCHING to 3-PARTITION it can readily be seen that they are gap-preserving (cf. [5]). Since these reductions also establish the strong NP-hardness of 3-PARTITION, the GAP-3-PARTITION problem has both claimed properties.

We show that the problem of finding a perfectly balanced partition with minimum cut size of a tree is APX-hard even if the maximum degree of the tree is at most 7. To prove this result we use a gap-preserving reduction from GAP-3-PARTITION in the following theorem.

Theorem 5.11. *Unless P=NP, there exists a constant ρ such that the k-BALANCED PARTITIONING problem on trees cannot be approximated in polynomial time within $\alpha = 1 + (1 - \rho^{-1})/24$, even if the maximum degree is at most 7.*

Proof. Consider an instance $\mathfrak{I}$ of GAP-3-PARTITION with polynomially bounded integers that are divisible by 12. Obviously all hardness properties are preserved by this restriction since GAP-3-PARTITION is strongly NP-hard and we may multiply each integer and the threshold parameter of an arbitrary instance by 12. As a consequence all integers are divisible by 4 and $s > 20$, which will become important later in the proof. For each a_i in $\mathfrak{I}$, construct a gadget H_i composed by a path on a_i vertices (called an a_i-path) connected to the root of a tree on s vertices (referred to as an s-tree). The root of the s-tree branches into four paths, three of them with $s/4$ vertices each, and one with $s/4 - 1$ vertices. Additionally the roots of the s-trees are connected in a path, as shown in Figure 5.3. We define Y to be the set of edges connecting different H_is and X the set of edges connecting an a_i-path with the corresponding s-tree in each H_i.

At a high level, we set out to prove that if all k integers in $\mathfrak{I}$ can be partitioned into triples that sum up to exactly s, then the constructed tree T can be split into $4k$ parts of a perfectly balanced partition with cut size $6k - 1$. If however at most k/ρ such triples can be found, T requires at least $(1 - \rho^{-1})k/4$

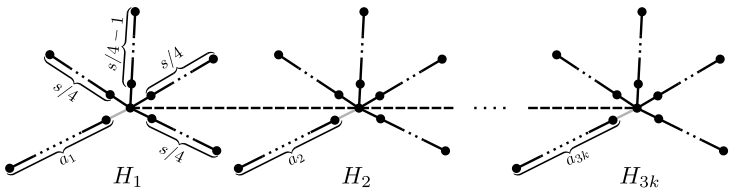


Figure 5.3: *Construction for Theorem 5.11. Each gadget H_i is composed by an a_i-path connected to the root of an s-tree through an edge from X (grey). Each s-tree branches into four paths of (almost) the same length. Two adjacent gadgets in a path are connected through the roots of their s-trees with an edge from Y (dashed).*

additional cut edges. This means that an algorithm computing an approximation within a factor smaller than $\frac{6k-1+(1-\rho^{-1})k/4}{6k-1}$ of the optimum cut size, can decide the GAP-3-PARTITION problem. Since the ratio $1+(1-\rho^{-1})/24$ is smaller, the theorem follows.

It is easy to see that if all k integers of $\mathfrak{I}$ can be partitioned into triples of size exactly s, cutting exactly the $6k-1$ edges in X and Y suffices to create a valid perfectly balanced partition into $4k$ parts.

It remains to be shown that $(1-\rho^{-1})k/4$ additional edges are required when the integers in $\mathfrak{I}$ can be partitioned into at most k/ρ triples of size exactly s. Let in this case Z^* be an optimal set of edges that cuts T into $4k$ parts of a perfectly balanced partition. We argue that by incrementally repositioning cut edges from the set $Z := Z^* \setminus (X \cup Y)$ to edges in $(X \cup Y) \setminus Z^*$, eventually all the edges in $X \cup Y$ will be cut. However, the following lemma implies that a constant fraction of the edges initially in Z will not be moved. We will then argue that the more triples of $\mathfrak{I}$ can not be packed into triples of size s, the more edges are left in Z. Thus the more edges must additionally have been in Z compared to those in $X \cup Y$. We rely on the following technical lemma which we will prove later.

Lemma 5.12. *If $s > 20$ then $|Z| \geq 2|(X \cup Y) \setminus Z^*|$.*

Consider the following algorithm **A** which repositions cut edges from a perfectly balanced partition into $4k$ parts. As long as there is an uncut edge $e \in X \cup Y$, **A** removes any cut edge left in Z and cuts e instead. At the end of the process, when all edges in $X \cup Y$ are cut, **A** removes the set of cut edges still left in Z denoted by Z'. Then $|Z'|$ is the number of additional edges cut in the case at most k/ρ triples that sum up to exactly s can be formed from the integers. When repositioning a cut edge from Z to $X \cup Y$, or when removing the edges in Z', **A** modifies the sizes of the sets in the partition induced by the cut set, and the balance might be lost. In particular, when a cut edge $e \in Z$ is removed, the algorithm will join the two connected components induced by the cut set and incident to e to form a single component. The algorithm will then include it in an arbitrary one of the sets that contained the two components. This changes the sizes of at most two sets in the partition. When a new cut is introduced by **A**, a component is split into two and the two newly created components are retained in the same set, thus no set size is changed.

By Lemma 5.12 there are at least as many edges in Z', as there are edges that are repositioned from Z to $X \cup Y$. Since each edge from Z repositioned by **A** causes at most two changes in set sizes, the total number of set size changes performed by **A** is at most $4|Z'|$.

When **A** terminates only edges from $X \cup Y$ are cut. Therefore the remaining connected components correspond to the $3k$ integers a_i of $\mathfrak{I}$ and $3k$ integers of size s. The integers in $\mathfrak{I}$ can be partitioned into at most k/ρ triples of size exactly s. Hence at least $(1 - \rho^{-1})k$ of the sets of the resulting partition do not have size exactly s. This means that **A** must have changed the size of at least $(1 - \rho^{-1})k$ sets, since it converted a perfectly balanced partition of T into a solution to `GAP-3-PARTITION` with at least $(1 - \rho^{-1})k$ unbalanced sets. This finally implies that $4|Z'| \geq (1 - \rho^{-1})k$, which concludes the proof since $|Z'|$ is the number of additional cuts required. □

To complete the above proof we still need to prove the technical lemma.

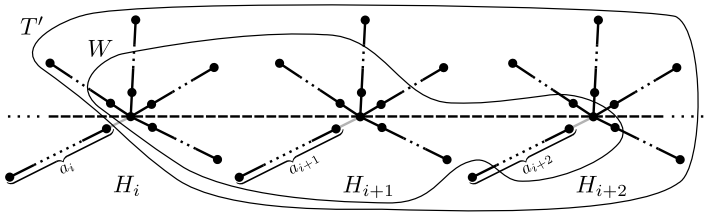


Figure 5.4: *Part of the construction of Lemma 5.12 with components W and T' highlighted*

Proof of Lemma 5.12. The crucial observation leading to the claim is that in any perfectly balanced partition of T into $4k$ parts, after removing the edges in Z^* from T every connected component has size at most s. We distinguish two cases. First consider an edge $e \in X \setminus Z^*$ such that all its adjacent edges in Y are also in Z^*. Let H_i be the gadget in which e is contained. The tree H_i has a total size of $s + a_i$. Since $a_i > s/4$ and each branch of the s-tree in H_i has at most $s/4$ vertices, there must be at least two edges from Z in H_i in order to cut away a_i vertices.

Otherwise consider a connected component W that contains uncut edges from $(X \cup Y) \setminus Z^*$. Let X' and Y' denote the edges in W from X and Y, respectively. Notice that $|Y'| \geq 1$ since the other case was considered above. Let T' be the subtree of T that results by extending W with all the a_i-paths incident to an edge in X' and all the s-trees incident to an edge in Y' (see Figure 5.4). Each branch of an s-tree and each a_i-path in T' has at least $s/4 - 1$ vertices. As $s > 20$, if at least 5 branches of s-trees or a_i-paths were fully included in W (before the extension to T') this connected component would contain more than s vertices. This is a contradiction since every connected component has size at most s. Therefore there are at most 4 such included branches. The branches that are not fully included in W but in T', each contain an edge from Z. Since T' contains at least $4(|Y'| + 1)$ s-tree branches and $|X'|$ a_i-paths, we can conclude that the number of edges from Z in T' is at least $|X'| + 4|Y'|$. Notice that $|Y'| \geq |X'| - 1$ since otherwise W would be disconnected.

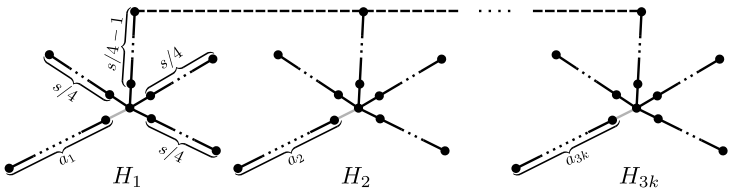


Figure 5.5: *Construction for Theorem 5.13. The used trees are almost the same as for Theorem 5.11 (Figure 5.3). The only difference is that the gadgets are connected in a path through one of their leaves instead of their roots.*

Using the fact that $|Y'| \geq 1$ we obtain

$$|X'| + 4|Y'| \geq 2|X'| - 1 + 3|Y'| \geq 2|X' \cup Y'|.$$

This proves our claim for the tree T'. Since the gadgets in any possible T' and the gadgets considered in the first case are pairwise disjoint, this concludes the proof. □

Using similar ideas as in the above proof, if we restrict the degree to be at most 5 we can still show that the problem remains NP-hard. For this we use a reduction from `3-PARTITION` together with a slightly different construction than the one shown in Figure 5.3. Instead of connecting the s-trees through their roots, the Y edges connect the leaves of the shortest branches of the s-trees (Figure 5.5). Connecting the gadgets through the leaves will only guarantee one additional cut edge if the `3-PARTITION` instance is not solvable. This is why this construction can not be used for the APX-hardness proof above, in which more edges were needed to be cut the less triples could be formed from the integers.

However for the trees as shown in Figure 5.5 it is possible to show that exactly the $6k - 1$ edges in X and Y are cut if all k integers in $\mathfrak{I}$ can be partitioned into triples of size exactly s, while otherwise at least $6k$ edges are cut. Since the `3-PARTITION` problem is strongly NP-hard this suffices to establish the following result, of which the proof can be found in the doctoral thesis of Luca Foschini.

Theorem 5.13. *The k-BALANCED PARTITIONING problem on trees has no polynomial time algorithm even if the maximum degree is at most* 5*, unless P=NP.*

5.5 Tightness of the Hardness Results

We presented a reduction framework which establishes the hardness of the k-BALANCED PARTITIONING problem for various graph classes. We specifically considered solid grid graphs and trees. For these we showed inapproximability results when perfectly balanced solutions are desired. Also we showed that, unless P=NP, there are no fast approximation algorithms that compute near-balanced partitions for arbitrary $\varepsilon > 0$. This is true even if the cut size may deviate from the optimum the more stringent the bound on the set sizes given by ε is. The latter bicriteria inapproximability proof was also considered for general graphs. These gave the first hardness results of this sort for the k-BALANCED PARTITIONING problem. At the same time we strengthened the known results for general graphs.

To meet the conditions identified by the reduction framework, we relied on the isoperimetric properties of grid graphs in order to show that a limited amount of edges can only cut off a bounded number of vertices. For trees we were able to harness high vertex degrees in order to meet the required conditions. This shows that even though trees and grid graphs have very different characteristics, their complexity with respect to the k-BALANCED PARTITIONING problem is similar. At the same time, by covering combinatorially simple but dissimilar graph classes, the ability of the framework to capture a fundamental trait of the problem is demonstrated. It remains to apply the framework to other graph classes that do neither share the isoperimetric properties of grids, nor the high vertex degrees of trees. One additional graph class that can be considered, but naturally shares the same respective properties with solid graph graphs, are simple shaped grid graphs containing holes. It is easy to see though, that for instance rectangular grid graphs having rectangular holes share the same hardness with solid grids. This can be established by

adding the respective edges that connect the upper left and right vertices of the gadgets in Figure 5.2 and slightly adapting the involved parameters.

All the hardness results derived in the first part of this chapter are asymptotically tight. For trees with arbitrary degrees this can be seen by considering a trivial approximation algorithm that simply cuts all edges and then greedily puts the vertices into sets of size at most $\lceil n/k \rceil$. Since a tree is connected and thus the optimum must cut at least one edge, this means that the cut size is approximated within a factor of n. Note that this algorithm shows the tightness of both Theorems 5.5 and 5.6, since for the latter it constitutes a fully polynomial time algorithm and a perfectly balanced partition is also near-balanced.

For connected grid graphs we can devise an approximation algorithm that computes perfectly balanced partitions using a greedy scheme similar to that used by MacGregor [47] for trees (cf. Section 5.1.2). We use Theorem 2.7 which says that any number of vertices can be cut from a grid graph using at most $\mathcal{O}(\sqrt{n})$ edges. In fact it is possible to devise a polynomial time algorithm that does this (cf. [18]). Hence we can greedily cut out the parts of a perfectly balanced partition repeatedly, using at most $\mathcal{O}(k\sqrt{n})$ edges in total. The optimal perfectly balanced solution has at least $\frac{n}{\lceil n/k \rceil} > k/2$ parts. Since the number of parts is integer and we assume the given grid graph to be connected, the optimal cut size is at least $k/2$. Therefore the approximation ratio of this algorithm is $\mathcal{O}(\sqrt{n})$, which shows the asymptotic tightness of both Theorems 5.8 and 5.9.

In addition to using the reduction framework we considered constant degree trees and showed that the complexity of the k-`BALANCED PARTITIONING` problem increases with the degree (Figure[5] 5.6). For trees of maximum degree 2, i.e. paths, the problem is trivial, while if the degree is 5 it becomes NP-hard. For degree 7 it is already hard to approximate within some constant, whereas for arbitrary degrees no approximation exists that is reasonable for practical applications. It remains open to generalise our results to show a tighter dependency of the

[5] This Figure can also be found in the doctoral thesis of Luca Foschini.

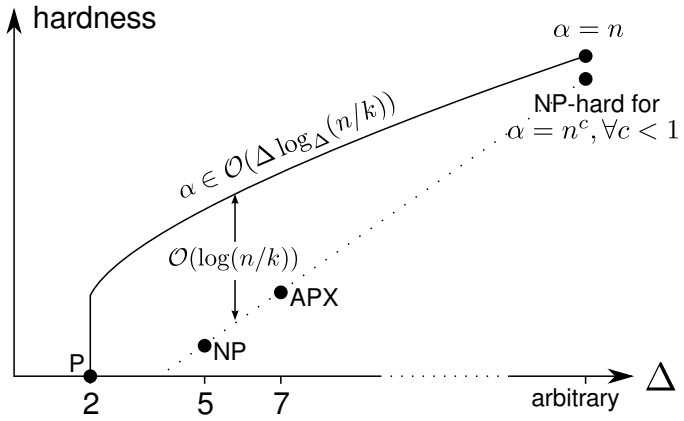


Figure 5.6: *The complexity results derived in this chapter for trees (big dots), depending on their maximum degree Δ. The hardness results are contrasted with the approximation algorithm by MacGregor [47]. For constant degree trees this algorithm entails a logarithmic gap between the complexity bounds.*

hardness on the maximum degree Δ. In addition, the possibility of an approximation algorithm for perfectly balanced partitions with a better ratio than $\alpha \in \mathcal{O}(\Delta \log_\Delta(n/k))$, as provided by the greedy scheme of MacGregor [47], remains open. In particular, for constant degree trees this algorithm gives a logarithmic approximation on the cut size. At the same time Theorem 5.11 does not rule out an algorithm that approximates the cut size by the factor $\alpha = 25/24$ if $\Delta \leq 7$. It also remains an open problem to determine the hardness of the problem when Δ equals 3 or 4. In particular it is not even known whether optimum solutions can be computed in polynomial time for perfect binary trees. Note however that the latter problem defines a *sparse language* since there is only one such tree per input length. As a result this means [50] that the problem on perfect binary trees can not be NP-hard, unless P=NP.

Chapter 6

Quo Vadis?

Throughout this thesis our aim was to give a theoretical foundation to understand the k-BALANCED PARTITIONING problem, with the practical application of distributing data in parallel-computing as background. Therefore we considered special cases of this problem that were tailored to represent practical needs as precisely as possible. In particular we modelled the graphs encountered in FEM simulations as grid graphs, and we were not satisfied with algorithms that were too slow or gave insufficient approximation guarantees. We explored how runtimes and approximation ratios can be traded for one another, and where the limits of finding such tradeoffs lie. Approaching the problem in these ways, it became apparent how its complexity boundaries depend on these tradeoffs.

For the special case when $k = 2$ (the BISECTION problem) the runtime can be traded for the solution quality. Even though we showed how the optimum on solid grid graphs can be computed in $\mathcal{O}(n^4)$ time, we also found much faster algorithms computing good bicriteria approximations in $\mathcal{O}(n^{1.5})$ time. The two criteria we considered are the balance of the set sizes and the cut size. For general k the runtime can also be traded for the solution quality, and furthermore this is necessary. When allowing the partitions to be a factor of 2 from perfectly balanced, we improved the

runtime for solid grids by allowing more edges to be cut. In particular it is known that the cut size can be approximated within a constant factor in $\widetilde{\mathcal{O}}(n^3)$ time, or $\widetilde{\mathcal{O}}(n^2)$ expected time. We improved the runtime to $\widetilde{\mathcal{O}}(n^{1.5})$ by allowing a logarithmic approximation ratio on the cut size. We further considered computing near-balanced partitions for arbitrary $\varepsilon > 0$, and showed that this can be achieved by allowing the runtime to increase when the limit on the balance becomes more stringent. However we also showed that the cut size does not need to increase at the same time. Furthermore we proved that it is not possible to trade the limit on the balance with the cut size, without also significantly increasing the runtime. Therefore the tradeoff between fast runtime and small set sizes is necessary.

Surprisingly often the above results for grid graphs were accompanied by results for trees, despite the fact that trees are very different from grids from a combinatorial point of view. On one hand, algorithms for trees were often the key to finding methods for grid graphs. On the other hand, the hardness of k-`BALANCED PARTITIONING` for trees gave an interesting contrast to the hardness for solid grids. Figure 6.1 summarises the best approximation factors achievable for both graph classes. For perfectly balanced solutions trees are harder to solve than grids, as witnessed by the results in Chapter 5. For near-balanced partitions however, the fact that trees do not have cycles makes it easier to devise approximations algorithms for them. This is capitalised in the dynamic program presented in Chapter 4.

Especially our bicriteria inapproximability results seem to suggest that no satisfying algorithms can be found, even for such simple graph classes as solid grid graphs or trees. Are the insights gained in this thesis, together with the related results, therefore all there is to say about the considered problem? In the following, we will summarise some possible ways how to circumvent the complexity results. For instance they do not rule out the existence of randomised algorithms that will perform well most of the times. Also it is not clear how the problem behaves on random inputs. That is, it may be possible that on a random graph the problem is considerably easier. Another alternative could be to examine k-`BALANCED PARTI-`

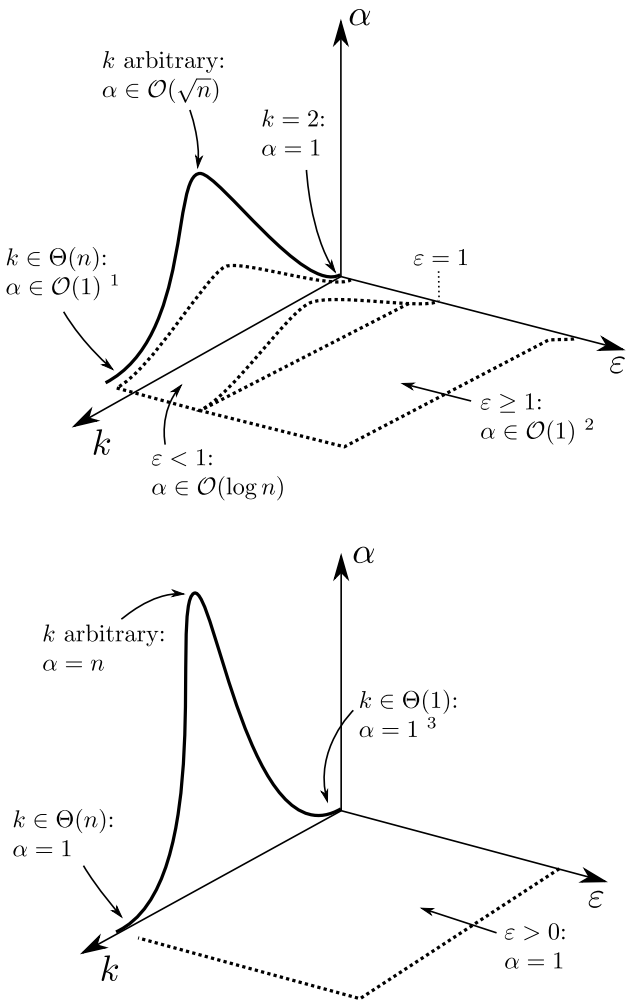


Figure 6.1: *Illustrations of best approximation factor α known against k and ε, for solid grid graphs (top) and trees (bottom). The plane (α, k) represents the case of perfectly balanced solutions. Here trees can be solved with better approximation quality than grids. For near-balanced partitions trees are better solvable. ([1] follows from [27], [2] from [20, 40], and [3] from [47]. All other results are proved in this thesis.)*

TIONING from a smoothed analysis point of view. It may be that the hard instances of the problem are isolated and therefore will hardly ever turn up in practical applications. Yet another possibility is to analyse the problem from a fixed parameter tractability point of view. That is, possibly there exists some parameter of the input for which a polynomial time algorithm can be found if the parameter is fixed to a constant. The only result known in this direction is for bounded tree-width graphs (cf. Section 4.4).

Alternatively a different model of the input instances could be chosen altogether. One possibility would be to consider graphs in which the optimal solution is guaranteed to consist of connected parts. This is justified by the observation that real-world instances typically do not encounter the "bottleneck" structure that the graphs used in our hardness proofs have. That is, they do not contain big components such as those acting as gadgets in the reductions, which are connected by only very few edges. Also the heuristics that are used to solve the problem in practice often compute solutions in which all cut out parts are connected. In general however the problem of finding a solution in which all parts are connected is not well-defined. Note that when considering bisections in planar graphs this condition means that the optimum consists of a single segment. There are for instance graphs in which a bisection cut out by a single segment does not exist (Figure 6.2).

In case it is guaranteed that for a given graph both parts of the optimal bisection are connected components, then for solid grid graphs the observations given in Chapter 2 can be used to compute an optimum solution in $\mathcal{O}(n^2)$ time. This is because according to Lemma 2.10 (page 30) all relevant segments can be enumerated in this time. Unfortunately the observations on near-optimal corner cuts from Chapter 3 can not be used to compute approximations in linear time by only considering straight and corner segments. This is because there are grid graphs in which any single such segment is an unsatisfying approximation for the optimum bisection (Figure 6.3).

Interestingly, for trees and arbitrary k the problem becomes trivial when all parts are connected in an optimum solution.

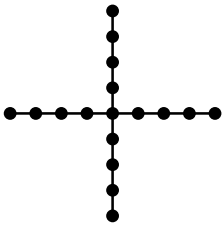


Figure 6.2: *A graph, which is a tree and a solid grid graph, in which no bisection has two connected parts. Note that the middle vertex must be contained in one of the parts. This makes it impossible for the other part to be connected because of the sizes of the spokes.*

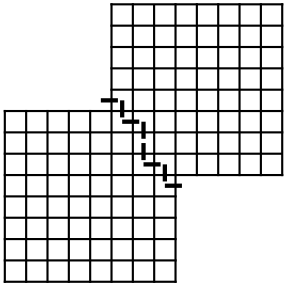


Figure 6.3: *A grid graph in which the optimum bisection (dashed line) has cut size* $\Theta(n^{1/4})$*. It is easy to see that any single straight segment contains* $\Omega(\sqrt{n})$ *edges. At the same time any single corner segment either has length* $\Omega(\sqrt{n})$ *or cuts out at least* $n/2 + \Omega(\sqrt{n})$ *vertices.*

A simple algorithm repeatedly cutting off the largest subtree of size at most $\lceil n/k \rceil$ will find the optimum in this case. This observation means that for trees the complexity of the k-`BALANCED PARTITIONING` problem results from the fact that it is hard to combine cut out components into parts. That this aspect of the problem is hard is not surprising since it corresponds to the NP-hard `BIN PACKING` problem [30]. By demanding that all parts are connected in an optimal solution the packing aspect of the problem is omitted. However for general graphs it remains unclear how this helps in deciding which edges to cut. Also for solid grid graphs it remains an open problem whether this helps algorithmically. Possibly, at least good approximations exist in this case.

Thus, there still seem to be possibilities to find algorithms with provably satisfying properties. Especially randomised schemes, fixed parameter algorithms, or even stricter assumptions on the inputs may provide algorithms that solve the problem contently. On the other hand, our hardness results entail that for arbitrary k it remains unexplained how deterministic methods can produce satisfying results. This is particularly true for graphs resulting from FEM simulations since we chose solid grid graphs as models, and these have a simpler structure than the graphs arising in practice. This is because solid grid graphs form a subclass of the (possibly) irregular three-dimensional topologies containing holes, that can be found in real-world applications.

Bibliography

[1] K. Andreev and H. Räcke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.

[2] P. Arbenz, G. van Lenthe, U. Mennel, R. Müller, and M. Sala. Multi-level μ-finite element analysis for human bone structures. In *Proceedings of the 8th Workshop on State-of-the-art in Scientific and Parallel Computing (PARA)*, pages 240–250, 2007.

[3] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. *Journal of Computer and System Sciences*, 58(1):193–210, 1999.

[4] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM*, 56(2):5, 2009.

[5] N. Bansal, D. Coppersmith, and B. Schieber. Minimizing setup and beam-on times in radiation therapy. In *Proceedings of the 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 27–38, 2006.

[6] Y. Bartal and S. Leonardi. On-line routing in all-optical networks. *Automata, Languages and Programming*, pages 516–526, 1997.

[7] S. N. Bhatt and S. S. Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Information Processing Letters*, 25(4):263–267, 1987.

[8] S.N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28(2):300–343, 1984.

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

[10] X. Cheng, D.-Z. Du, J.-M. Kim, and L. Ruan. Optimal rectangular partitions. In *Handbook of Combinatorial Optimization*, pages 313–327. Springer US, 2005.

[11] C. Chevalier and F. Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering. *Parallel Computing*, 34(68):318–331, 2008.

[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT press, 2001.

[13] D. Delling, A. Goldberg, T. Pajor, and R. Werneck. Customizable route planning. *Experimental Algorithms*, pages 376–387, 2011.

[14] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science Engineering*, 4(2):90–96, 2002.

[15] J. Díaz, J. Petit, and M. J. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

[16] J. Díaz, M. J. Serna, and J. Torán. Parallel approximation schemes for problems on planar graphs. *Acta Informatica*, 33(4):387–408, 1996.

[17] R. Diestel, T. R. Jensen, K. Y. Gorbunov, and C. Thomassen. Highly connected sets and the excluded grid theorem. *Journal of Combinatorial Theory, Series B*, 75(1):61–73, 1999.

[18] K. Diks, H. N. Djidjev, O. Sykora, and I. Vrto. Edge separators of planar and outerplanar graphs with applications. *Journal of Algorithms*, 14(2):258 – 279, 1993.

[19] H.C. Elman, D.J. Silvester, and A.J. Wathen. *Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics*. Oxford University Press, USA, 2005.

[20] G. Even, J.S. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. *SIAM Journal on Computing*, 28(6):2187–2214, 1999.

[21] A. E. Feldmann. Fast balanced partitioning is hard, even on grids and trees. *ArXiv e-prints*, (arXiv:1111.6745), 2011.

[22] A. E. Feldmann, S. Das, and P. Widmayer. Simple cuts are fast and good: Optimum right-angled cuts in solid grids. In *Proceedings of the 4th Annual International Conference on Combinatorial Optimization and Applications (COCOA)*, pages 11–20, 2010.

[23] A. E. Feldmann, S. Das, and P. Widmayer. Restricted cuts for bisections in solid grids: A proof via polygons. In *Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 143–154, 2011.

[24] A. E. Feldmann and L. Foschini. Balanced partitions of trees and applications. In *29th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 100–111, 2012.

[25] A. E. Feldmann and P. Widmayer. An $O(n^4)$ time algorithm to compute the bisection width of solid grid graphs. In *Proceedings of the 19th Annual European Symposium on Algorithms (ESA)*, pages 143–154, 2011.

[26] T. Feo, O. Goldschmidt, and M. Khellaf. One-half approximation algorithms for the k-partition problem. *Operations Research*, 40:170–173, 1992.

[27] T.A. Feo and M. Khellaf. A class of bounded approximation algorithms for graph partitioning. *Networks*, 20(2):181–195, 1990.

[28] D. Fleischer and U. Brandes. Vertex bisection is hard, too. *Journal of Graph Algorithms and Applications*, 13(2):119–131, April 2009.

[29] Z. Galil. Efficient algorithms for finding maximum matchings in graphs. *ACM Computing Surveys*, 18:23–38, March 1986.

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

[31] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[32] M. Goldberg and Z. Miller. A parallel algorithm for bisection width in trees. *Computers & Mathematics with Applications*, 15(4):259–266, 1988.

[33] D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

[34] G. Karypis and V. Kumar. METIS-unstructured graph partitioning and sparse matrix ordering system, version 2.0. Technical report, University of Minnesota, 1995.

[35] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *Proceedings of the 36th ACM/IEEE Conference on Design Automation*, pages 343–348, 1999.

[36] C. Kenyon and E. Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25:645–656, November 2000.

[37] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, 1970.

[38] S. A. Khot and N. K. Vishnoi. The Unique Games Conjecture, integrality gap for cut problems and embeddability of

negative type metrics into ℓ_1. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 53–62, 2005.

[39] D. G. Kirkpatrick and P. Hell. On the completeness of a generalized matching problem. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 240–245, 1978.

[40] P. Klein, S.A. Plotkin, and S. Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 682–690, 1993.

[41] E. Koutsoupias, C. H. Papadimitriou, and M. Sideri. On the optimal bisection of a polygon. *ORSA Journal on Computing*, 4(4):435–438, 1992.

[42] R. Krauthgamer, J. Naor, and R. Schwartz. Partitioning graphs into balanced components. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 942–949, 2009.

[43] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003.

[44] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

[45] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[46] R.J. Lipton and R.E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9:615–627, 1980.

[47] R. M. MacGregor. *On Partitioning a Graph: a Theoretical and Empirical Study.* PhD thesis, University of California, Berkeley, 1978.

[48] A. Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 245 –254, 2010.

[49] B.M. Maggs, F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Exploiting locality for data management in systems of limited bandwidth. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 284–293. IEEE, 1997.

[50] S. R. Mahaney. Sparse complete sets for NP: Solution of a conjecture of berman and hartmanis. *Journal of Computer and System Sciences*, 25(2):130–143, 1982.

[51] D.W. Matula and F. Shahrokhi. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Mathematics*, 27(1-2):113–123, 1990.

[52] K. Nakano. Linear layouts of generalized hypercubes. In *Proceedings of the 19th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 364–375. Springer, 1993.

[53] L. Palios. Upper and lower bounds for optimal tree partitions. Unpublished manuscript.

[54] C. Papadimitriou and M. Sideri. The bisection width of grid graphs. *Theory of Computing Systems*, 29:97–110, 1996.

[55] J. K. Park and C. A. Phillips. Finding minimum-quotient cuts in planar graphs. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 766–775, 1993.

[56] E. Petrank. The hardness of approximation: Gap location. *Computational Complexity*, 4(2):133–157, 1994.

[57] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, 2008.

[58] L. Råde and B. Westergren. *Mathematics Handbook for Science and Engineering*. Springer Verlag, 2004.

[59] V. Ramachandran. Finding a minimum feedback arc set in reducible flow graphs. *Journal of Algorithms*, 9(3):299 – 313, 1988.

[60] S. Rao. Faster algorithms for finding small edge cuts in planar graphs. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 229–240, 1992.

[61] J. Rolim, O. Sỳkora, and I. Vrto. Optimal cutwidths and bisection widths of 2- and 3-dimensional meshes. In *Proceedings of the 21st International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 252–264, 1995.

[62] H.R. Schwarz. *Methode der finiten Elemente: eine Einführung unter besonderer Berücksichtigung der Rechenpraxis*. BG Teubner, 1991.

[63] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[64] D.B. Shmoys. Cut problems and their application to divide-and-conquer. In D. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 192–235. PWS Publishing Co., 1996.

[65] H. D. Simon and S. H. Teng. How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.

[66] K. Soumyanath and J. S. Deogun. On the bisection width of partial k-trees. In *Proceedings of the 20th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, volume 74 of *Congressus Numerantium*, pages 25–37, 1990.

[67] G. Strang. Maximum area with Minkowski measures of perimeter. In *Proceedings of the Royal Society of Edinburgh: Section A Mathematics*, volume 138, pages 189–199, 2008.

[68] S. Ulmer. Giganten für die Zukunft trimmen. *ETH Globe*, (4):9–15, November 2009.

[69] V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[70] D. Wagner and F. Wagner. Between min cut and graph bisection. In *Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 744–750, 1993.

[71] D.B. West. *Introduction to Graph Theory*, volume 2. Prentice Hall Upper Saddle River, NJ.:, 2001.

[72] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

Glossary

Graphs

G	graph or grid graph
v, w, u	vertices
V	set of vertices
n	number of vertices in a graph
e	edge
ω	edge weight
E, X, Y, Z	sets of edges
T	tree
r	root of a tree
l	number of leaves in a tree
D	dual graph
f	face
p, q	paths
Δ	maximum degree
W	connected component
x	size of a connected component
$\mathcal{W}$	set of connected components
$\mathfrak{W}$	set of sets of connected components

Partitions and Cuts in Graphs

$\mathcal{V}$	vertex partition
$\mathcal{L}$	leaf partition
k	number of parts in a partition

m	number of cut out vertices
$A, B, P, \overline{P}$	parts
$\mathcal{P}$	set of parts
C	cut size
s, t, u, r	segments
b	bar segment
f	bend
S, T	sets of segments
$\mathcal{T}$	segment family
$\mathcal{S}$	all relevant segments of a solid grid graph
$\mathcal{C}$	straight and corner segments

Polygons

$\mathscr{P}, \mathscr{Q}, \mathscr{D}, \mathscr{L}, \mathscr{X}, \mathscr{Y}$	polygons and sub-polygons
$\mathscr{C}, \mathscr{V}, \overline{\mathscr{V}}$	corridor and parts of corridors
$\mathscr{R}$	rectangle
$\mathscr{S}$	square
$\mathscr{T}$	tail
$\mathscr{Z}$	visible area from a line
β, γ	boundaries
p, q, r	points
P	set of points
a, b, d	sizes of areas
h	height
w	width
x, y	coordinates
z	coordinate offset
Q	set of unit squares
U	set of unit length lines
I, J, K	intervals on axes

Cuts in Polygons

m	size of cut out area
$\mathscr{A}, \mathscr{B}, \mathscr{C}, \overline{\mathscr{C}}$	parts in a polygon
C	cut size

$\lambda, \mu, \delta, \rho, \sigma, \tau$	lines
L, M, K	sets of lines
Λ, Δ, Ξ	virtual lines
l	length

Algorithms

$\mathbf{A}$	algorithm
α	approximation factor on the cut size
ε	approximation factor on the balance
β	approximation factor
$\mathcal{K}$	IFS covering set
b	balance of an edge separator
d	difference between balances
μ	maximum size of a part
$\vec{g}$	signature
$\vec{e}$	unit vector
σ	number of signatures
t	runtime parameter & size of signature
τ	number of execution steps
L	set of vertices in subtrees
N	number of vertices in subtrees
$\mathcal{F}$	lower frontier
$\mathfrak{I}$	instance
$\mathfrak{e}$	element of bin packing instance
ϑ	number of bins
λ	weight of decomposition tree

Reductions

ρ	constant approximation factor
s	threshold
a_i	integer
I	set of integers
$\mathcal{I}$	set of sets of integers
H	gadget
g	number of edges connecting gadgets
μ	parameter determining the size of a gadget

Miscellaneous

c, d	constants
i, j, l, r	indices
I, J	sets of indices
π	polynomial
f, φ	functions

Index