Approximation of the minimum bisection and the hardware–software partitioning problem

Thesis

András Orbán

Supervisor: András Benczúr Jr.

Department of Operations Research
Eötvös Loránd University
Faculty of Science

2004

Contents

Chapter 1

Introduction

1.1 Motivation

Today's computer systems typically consist of both hardware and software components. For instance in an embedded signal processing application it is common to use both application–specific hardware accelerator circuits and general–purpose, programmable units with the appropriate software [3].

This is beneficial since application–specific hardware is usually much faster than software, and also more power–efficient, but it is also significantly more expensive. Software on the other hand is cheaper to create and to maintain, but slow, and general–purpose processors consume much power. Hence, performance or power critical components of the system should be realized in hardware, and non–critical components in software. This way, an optimal trade–off between cost, power and performance can be achieved.

One of the most crucial steps in the design of such systems is *partitioning*, *i.e.* deciding which components of the system should be realized in hardware and which ones in software. Clearly, this is the step in which the above–mentioned optimal trade–off has to be found. Therefore, partitioning has dramatic impact on the cost and performance of the whole system [18]. The complexity of partitioning arises because conflicting requirements on performance, power, cost, chip size, *etc.* have to be taken into account.

There are several versions of the partitioning problem, we now deal with the one defined in [18]. An informal definition follows. (See Chapter 2 for an exact definition.) The system to be partitioned is modelled by a *communication graph*, the nodes of which are the components of the system that have to be mapped to either hardware or software, and the edges represent communication between the components. Unlike in most previous models for partitioning (a good survey about partitioning models is [17]), it is not assumed that this graph is acyclic in the directed sense. The edges are not even directed, because they do not represent data flow or dependency. Rather, their role is the following: if two communicating components are mapped to different contexts (*i.e.* one to hardware and the other to software, or vice versa), then their communication incurs a communication penalty, the value of which is given for each edge as an edge cost. This is assumed to be independent of the direction of the communication (whether from hardware to software or vice versa). If the communication does not cross the hardware/software boundary, it is neglected.

Besides the edge costs, each vertex is assigned two cost values called hardware cost and software cost. If a given vertex is decided to be in hardware, then its hardware cost is considered, otherwise its software cost. We do not impose any explicit restrictions on the semantics of hardware costs and software costs; they can represent any cost metrics, like execution time, size, or power consumption. Likewise, no explicit restriction is imposed on the semantics of communication costs. Nor do we impose explicit restrictions on the granularity of partitioning (*i.e.* whether nodes represent instructions, basic blocks, or procedures). However, we assume that the total hardware cost with respect to a partition can be calculated as the sum of the hardware costs of the nodes that are in hardware, and similarly, the software cost with respect to a partition can be calculated as the sum of the software costs of the nodes that are in software, just as the communication cost with respect to a partition, which is the sum of the edge costs of those edges that cross the boundary between hardware and software.

Several optimization (and decision) problems can be defined on this model, most of which are known to be $\mathcal{NP}$–hard ($\mathcal{NP}$–complete). See Section 2.2 for the exact definitions and the proofs. However, to our best knowledge currently no approximation factor concerning these problems is known. The aim of this thesis is to develop approximation algorithms for the hardware–software partitioning problem.

1.2 Divide–and–conquer approach

In the design of approximation algorithms for several hard graph–theoretic optimization problems the *divide–and–conquer* approach is beneficial. Examples include the minimum feedback arc set problem, the minimum cut linear arrangement problem, the storage–time product problem, perfect elimination ordering etc. See [13] for a good summary. The main idea of this approach is to partition the input graph into two ore more roughly equal parts that can be solved independently and to assemble a solution for the original problem of the solution of the parts.

Divide–and–conquer methods motivate the search for a *balanced cut* (there is an upper limit on the size of the parts) or specially an exact *bisection* (both parts are of equal size) in a graph. The objective is to minimize the number (or weight) of edges cut. Another formulation of the balanced cut problem is the so called *minimum ratio cut*, where there is no restriction on the size of the parts, rather the objective is to minimize the ratio of the cut and the smaller part. Unfortunately all of these problems are $\mathcal{NP}$–hard [10].

The hardware–software partitioning problem is closely related to the minimum bisection problem as will be shown in Chapter 4. This motivates us to examine the approximation algorithms developed for the minimum bisection problem.

In pioneering works, Leighton and Rao [15, 16] give an $\mathcal{O}(\log n)$–approximation algorithm for the minimum ratio cut, which implies a polylogarithmic approximation ratio on several related problems. However it does not yield an approximation algorithm for the minimum balanced cut problem, only a pseudo–approximation factor can be guaranteed. The first real approximation algorithm for the minimum bisection is by Saran and Vazirani [20], who gave an $n/2$–approximation algorithm based on Gomory–Hu cuts. This has been radically improved by Feige and Krauthgamer reaching an approximation factor of $\mathcal{O}(\sqrt{n}\log n)$ in [7]. In [6] they significantly improved their own result, presenting an $\mathcal{O}(\log^2 n)$–approximation algorithm based on the minimum ratio cut approximation of Leighton and Rao. Currently this is the best known approximation factor for general graphs, although there is no hardness evidence whether a constant approximation factor could be reached. However, Bui and Jones presented an interesting $\mathcal{NP}$–hardness result. In [5] they prove that it is $\mathcal{NP}$–hard to approximate the minimum bisection within an $n^{2-\varepsilon}$ *additive* error. For restricted classes of graphs better approximation ratios can be reached. Arora et al. gave a polynomial time approximation scheme (PTAS) for the minimum bisection on (everywhere) dense graphs [4]. Garg et al. on the other hand deals with sparse graphs: they show in [11] that the 2/3–balanced cut can be approximated within twice the optimal on planar graphs. However this approach does not extend to bisection. More generally if the graph excludes any given minor, the minimum ratio cut can be approximated within a constant ratio [14], implying an $\mathcal{O}(\log n)$–approximation on the minimum bisection due to [6].

1.3 Organization

This work is organized as follows. In Chapter 2 we formally define the problems we are dealing with, Chapter 3 introduces the known approximation algorithms for the minimum bisection. Here we only outline the basic ideas and state the main results, for the proofs we generally refer to the original article. Based on the known approximation algorithms we present a polylogarithmic approximation algorithm for (some versions of) the hardware–software partitioning problem in Chapter 4, while Chapter 5 concludes the thesis.

Chapter 2

Problem definition

2.1 Minimum bisection problem

Given an undirected, simple graph $G(V, E)$, each $S \subset V, S \neq \emptyset, V$ defines the cut $(S, V \setminus S)$, *i.e.* those edges with one endpoint in S and the other in $S \setminus V$. The number of crossing edges of this cut is denoted by $e(S, V \setminus S)$.

Definition 2.1. *Fixing a parameter $\frac{1}{2} \leq \alpha < 1$, the* minimum α–balanced cut problem *is to find a cut $(S, V \setminus S)$, so that $|S| \leq \alpha|V|$ and $|V \setminus S| \leq \alpha|V|$ and $e(S, V \setminus S)$ is minimized.*

Definition 2.2. *Assuming n to be even, the* minimum bisection problem *is to find a cut where $|S| = \frac{n}{2}$ with minimum edges cut. It can be regarded as the special case of Definition 2.1 with $\alpha = \frac{1}{2}$. We will often use the notation (W, B) to refer to a (white–black) bisection. The value of the optimal bisection is denoted by b.*

Note that in the basic version of the minimum bisection problem no edge costs or vertex costs are defined. Both the edge–weighted and the vertex–weighted (or combined) problems can be defined.

Definition 2.3. *In the* edge–weighted *version of the bisection problem the edges are assigned an arbitrary non–negative cost $c : E \to \mathbb{R}^+$, and the objective is to minimize the edge–weights cut over all bisections.*

Definition 2.4. *In the* vertex–weighted *version of the bisection problem the vertices are assigned non–negative integer weights $w : V \to \mathbb{N}$, bounded by a polynomial of n, n^c for some c. S is a bisection if $w(S) = \frac{\sum_V w_i}{2}$. The objective is again to minimize the number of edges cut.*

Remark 2.1. *The polynomial bound on the vertex weights is necessarily, otherwise it would be $\mathcal{NP}$–complete just to decide whether a graph has a bisection or not. (It would be equivalent to the subset–sum problem.)*

Another approach to achieve a balanced cut is to relax the bound on the sizes of the parts and build it into the objective function. The most famous and well studied formulation of this concept is the minimal ratio cut.

Definition 2.5. *The* ratio *of the cut $(S, V \setminus S)$ is*

$$r(S) := \frac{e(S, V \setminus S)}{\min\{|S|, |V \setminus S|\}}$$

The minimum ratio cut *problem consists of finding the cut with minimum ratio among all the cuts. The* ratio of a cut towards S *is denoted by* $r'(S) := \frac{e(S,V \setminus S)}{|S|}$.

Throughout this thesis if we use the notation $f(X)$ where $X = \{x_1, \dots, x_l\}$ is a set of l elements and f is a function on the elements of X, then $f(X)$ means $\sum_{i=1}^{l} f(x_i)$. For example, $c(B), B \subseteq E$ or $c(S, V \setminus S)$ denotes the total weight of edges in B or in the cut $(S, V \setminus S)$, respectively; $w(S), S \subseteq V$ denotes the sum of vertex weights in S.

2.2 Partitioning problem

An undirected simple graph $G = (V, E)$, $V = \{v_1, \dots, v_n\}$, $s, h : V \to \mathbb{R}^+$ and $c : E \to \mathbb{R}^+$ are given. $s(v_i)$ (or s_i) and $h_(v_i)$ (or h_i) denote the software and hardware cost of node v_i, respectively, while $c(v_i, v_j)$ denotes the communication cost between v_i and v_j that occurs only if this edge is cut, *i.e.* if v_i and v_j are in different contexts (HW or SW). We denote the number of nodes by n, the number of edges by m.

P is called a (hardware–software, HW–SW) partition of G if it is a bipartition of V into $V = V_H \uplus V_S$. The hardware cost of P is: $H_P := h(V_H)$; the software cost of P is: $S_P := s(V_S)$; the communication cost is $C_P := c(V_H, V_S)$. Often the software cost denotes the execution time of the software unit (since this is the most critical factor), and the communication cost denotes the communication delay induced on the edges. As a consequence, sometimes it makes sense to add them to get the overall system execution time, $R_P := S_P + C_P$ (the hardware is much faster, thus its execution time can be neglected.) The following optimization and decision problems can be defined (G, h, s, c are given in all problems):

Definition 2.6. *In the* PART1 *problem α, β, γ non–negative constants are given. The goal is to minimize $\alpha S_P + \beta H_P + \gamma C_P$ over the choice of P.*

Definition 2.7. *In the* PART2 *problem H_0, $R_0 \in \mathbb{R}^+$ are given. It should be decided whether there is a P HW–SW partition so that $H_P \le H_0$ and $R_P \le R_0$?*

Definition 2.8. *In the* PART3 *problem $H_0 \in \mathbb{R}^+$ is given. The goal is to find a P HW–SW partition so that $H_P \le H_0$ and R_P is minimal.*

Definition 2.9. *In the* PART4 *problem $R_0 \in \mathbb{R}^+$ is given. The goal is to find a P HW-SW partition so that $R_P \le R_0$ and H_P is minimal.*

2.3 NP-hardness results

Theorem 2.1. *All versions of the balanced cut problem and the graph bisection problem are $\mathcal{NP}$–hard.*

Proof. The proof can be found in [10]. □

Theorem 2.2. *The min ratio cut problem is $\mathcal{NP}$–hard.*

Proof. The proof can be found in [21]. □

The best approximation algorithm is published by Leighton and Rao in their famous articles [15, 16].

Theorem 2.3 (Leighton and Rao, 1988). *The minimum ratio cut problem can be approximated within a factor of $\mathcal{O}(\log n)$.* □

The hardness of the partitioning problems is characterized by the following theorems, which are our own results.

Theorem 2.4. PART1 *can be solved in polynomial time.*

Proof. It can be reduced to the minimum weighted $s-t$ cut problem in an undirected graph, which can be solved using the max–flow–min–cut theorem of Ford–Fulkerson [8]. For the construction and the detailed proof see [2, 22] $\square$

Theorem 2.5. PART2 *is $\mathcal{NP}$-complete even if no edges are present.*

Proof. PART2$\in \mathcal{NP}$, since P is a good proof for that.

To prove the $\mathcal{NP}$-hardness, we reduce the KNAPSACK problem [19] to PART2. Let an instance of the KNAPSACK problem be given. (There are n objects, the weights of the objects are denoted by w_i, the price of the objects by p_i, the weight limit by L and the price limit by K. The task is to decide, whether there is a subset X of objects, so that $w(X) \leq L$ and $p(X) \geq K$.) We define a graph to that as follows: $V = \{v_1, \ldots, v_n\}$, $E = \{\}$. Let $h_i = p_i$, $s_i = w_i$. (Since E is empty, there is no need to define c.) Introducing $A := p(V)$, let $R_0 = L$, $H_0 = A - K$.

Now we solve PART2 with these parameters. We state that it has a solution iff the given KNAPSACK problem has a solution.

Assuming that PART2 has a solution: $V = V_H \uplus V_S$. It means that

$$w(V_S) \leq L \tag{2.1}$$

and

$$p(V_H) \leq A - K = p(V) - K$$

the last one can also be formulated as:

$$K \leq p(V) - p(V_H) = p(V_S) \tag{2.2}$$

(2.1) and (2.2) proves that $X = V_S$ is a solution of the original KNAPSACK problem.

Let now assume that X solves the KNAPSACK problem. Therefore:

$$s(X) = w(X) \leq L = R_0 \tag{2.3}$$

and

$$p(X) \geq K = A - H_0 = p(V) - H_0$$

that is

$$H_0 \geq p(V) - p(X) = p(V \setminus X) = h(V \setminus X) \tag{2.4}$$

(2.3) and (2.4) verifies that $V = (V \setminus X) \uplus X$ solves PART2. $\square$

The previous theorem proves the $\mathcal{NP}$–completeness only in the weak sense. However, PART2 is $\mathcal{NP}$–hard in the strong sense as well, *i.e.* if the weights of the nodes must be polynomial in n.

Theorem 2.6. PART2 *is $\mathcal{NP}$-complete even if the vertex and edge weights are polynomial in n.*

Proof. We reduce the decision version of the minimum bisection problem as defined in Definition 2.2, which is known to be $\mathcal{NP}$–complete [10], to PART2.

Given an instance of the minimum bisection problem on $G(V, E)$ with n vertices, where n is even, m edges and a limit K, our goal is to find a cut (W, B), for which $|W| = |B| = \frac{n}{2}$ and the cutsize is at most K ($K \leq m$).

Now associate to it the following instance of the PART2 problem. Let $h(v_i) = s(v_i) = 1$ for each $v_i \in V$ and let $c(v_i, v_j) = \frac{1}{m+1}$ for each $(i, j) \in E$. Define $H_0 := \frac{n}{2}$ and $R_0 := \frac{n}{2} + \frac{K}{m+1}$. Clearly this instance has polynomial weights in n.

We claim that the two problems have identical solution sets. Indeed, if (W, B) is a solution for the bisection problem ($|W| = |B| = \frac{n}{2}$ and $e(W, B) \leq K$), then the same (W, B) solves PART2 as well, since $h(W) = |W| \leq H_0$ and $s(B) + c(W, B) = |B| + \frac{1}{m+1}e(W, B) \leq \frac{n}{2} + \frac{K}{m+1} = R_0$.

Vice versa, if the partition (V_H, V_S) is a feasible solution of PART2, then $h(V_H) = |V_H| \leq \frac{n}{2}$ and $s(V_S) + c(V_H, V_S) \leq \frac{n}{2} + \frac{K}{m+1} < \frac{n}{2} + 1$, thus $s(V_S) = |V_S| \leq \frac{n}{2}$, as it is an integer. As both sides of the partition (V_H, V_S) are not larger than $\frac{n}{2}$, $|V_H| = |V_S| = \frac{n}{2}$ must hold. This also implies—using again the condition for the overall execution time—that $c(V_H, V_S) \leq \frac{K}{m+1}$, hence $e(V_H, V_S) \leq K$. So (V_H, V_S) is indeed a solution for the bisection problem as well. □

Similar proofs can be established for PART3 and PART4.

Theorem 2.7. PART3 *and* PART4 *are $\mathcal{NP}$-hard in the strong sense.* □

Chapter 3

Approximation of the minimum bisection

In this chapter known approximation algorithms for the minimum bisection problem are presented. These algorithms are of independent interest, while some of them can also be used to develop approximation algorithms for the hardware–software partitioning problem. Section 3.1 introduces the first known approximation algorithm to the problem based on [20]. The most advanced works of Feige and Krauthgamer [7, 6] are shown in Section 3.2. For dense instances of the bisection problem a polynomial time approximation scheme is given in Section 3.3 based on [4], while an interesting theorem of additive approximation published in [5] is presented in Section 3.4.

3.1 Simple approximation algorithm

In this section we give an $\frac{n}{2}$–approximation algorithm to the vertex–weighted version (see Definition 2.4) of the minimum bisection.

The steps of the algorithm can be seen in Algorithm 1. In Step 1 we use the classical result of Gomory and Hu [12], namely that there is a set of $n-1$ cuts in G such that for each pair of vertices $u, v \in V$ the set contains a minimum weight separating cut between u and v and these cuts can be calculated using only $n-1$ flow computations.

Algorithm 1 Simple approximation algorithm for the minimum bisection problem

```
1. Find a set of Gomory–Hu cuts in $G$.
2. Sort these cuts by increasing weights, obtaining $g_1, \ldots, g_{n-1}$.
3. Find the minimum $i$ such that the connected components of $G' := (V, E \setminus (g_1 \cup \ldots \cup g_i))$ can
   be partitioned into two sets of size $\frac{n}{2}$.
```

It is known that there is a pseudo–polynomial time algorithm for the problem of dividing n into equal parts [9], captured by the following lemma.

Lemma 3.1. *Given the numbers $a_1, \ldots, a_n$ with $\sum_{i=1}^{n} a_i = K$ there is a pseudo–polynomial algorithm, which can decide in $\mathcal{O}(nK)$ time whether there is a set of indexes $I \subseteq \{1, \ldots, n\}$ so that $\sum_I a_i = \frac{K}{2}$.* □

Since the sum of the sizes of the components in Step 3 of Algorithm 1 is n, according to Lemma 3.1 this step takes $\mathcal{O}(n^2)$ time for each $i \in \{1, \ldots, n-1\}$, thus all together it takes $\mathcal{O}(n^3)$ time. Hence Algorithm 1 is indeed polynomial and its running time is dominated by Step 1.

Before proving the approximation factor of the algorithm some preparation is needed. Let $B := g_1 \cup \ldots \cup g_i$ be the set of edges found by the algorithm. Among the cuts $g_1, \ldots, g_i$, pick $g_j, j \leq i$ if it is not contained in $g_1 \cup \ldots \cup g_{j-1}$. Let $b_1, \ldots, b_l$ be the cuts picked in this manner. Clearly $B = b_1 \cup \ldots \cup b_l$. Let $d_1, d_2, \ldots$ be an enumeration of all cuts in G ordered by increasing weight. For any cut d in G denote by $index(d)$ its index in this enumeration.

Definition 3.1. *The cuts $b_1, \ldots, b_l$ are said to be* consistent *with such an enumeration $d_1, d_2, \ldots$ if they appear in this order in the enumeration.*

We will use the following important property.

Definition 3.2. *Let $b_1, \ldots, b_p$ be a set of cuts in G, sorted by increasing weight. Pick any enumeration of all cuts in G, $d_1, d_2, \ldots$ that is consistent with $b_1, \ldots, b_p$. The cuts $b_1, \ldots, b_p$ satisfy the* union property *if the union of the cuts in any initial segment of $d_1, d_2, \ldots$ is equal to the union of all cuts b_j contained in this initial segment. More formally, pick any consistent enumeration of all cuts in G, let j be any index, $1 \leq j \leq index(b_p)$, and let b_q be the last cut in the sorted order having index at most j. Then, $d_1 \cup \ldots \cup d_j = b_1 \cup \ldots \cup b_q$.*

Lemma 3.2. *The cuts $b_1, \ldots, b_l$ satisfy the union property.* □

The following rather technical lemma says that from $\frac{n}{2} + 1$ components the bisection can surely be assembled, *i.e.* no more than $\frac{n}{2}$ cuts are needed.

Lemma 3.3. *Let n be an even integer, and let $a_1, \ldots, a_{\frac{n}{2}+1}$ be positive integers such that*

$$\sum_{i=1}^{\frac{n}{2}+1} a_i = n$$

Then there is a set of indexes $I \subseteq \{1, \ldots, \frac{n}{2} + 1\}$ so that $\sum_I a_i = \frac{n}{2}$. □

Let d_k be the first (in the enumeration $d_1, d_2, \ldots$) optimal bisection.

Proposition 3.1. *$index(b_l) \leq k$*

Proof. Assuming to the contrary that $index(b_l) > k$ then choosing $j = k$ in the Definition 3.2 and using that according to Lemma 3.2 the cuts $b_1, \ldots, b_l$ satisfy the union property yield that

$$d_1 \cup \ldots \cup d_k = b_1 \cup \ldots \cup \ b_q \tag{3.1}$$

for a certain $q < l$. Since the components of $G^* = (V, E \setminus C)$ can certainly be partitioned into equal sized sets, for any set C containing d_k imply that the left–hand side of Equation (3.1) can be partitioned into equal sized sets, hence so is the right–hand size. However, according to the selection of the cuts $b_1, \ldots, b_l$ this would imply that $q = l$, which is a contradiction. □

Theorem 3.1 (Saran and Vazirani, 1995). *B approximates the minimum bisection by a factor of $\frac{n}{2}$,* i.e. *$c(B) \leq \frac{n}{2} c(d_k)$*

Proof. It follows from Proposition 3.1 that $c(b_j) \leq c(d_k)$ for each $1 \leq j \leq l$. To prove the desired approximation factor, l must be constrained. Lemma 3.3 implies that $l \leq \frac{n}{2}$ finishing the proof. □

3.2 Polylogarithmic approximation

In this section the $\mathcal{O}(\log^2 n)$–approximation algorithm for the basic bisection problem (recall Definition 2.2) from Fiege and Krauthgamer [6] is presented. To our knowledge this is currently the best known approximation algorithm to the problem. The basic algorithm can be extended in several ways as shown in Section 3.2.5. Both the basic algorithm and the extensions are of special importance to us, because they can be used to develop an approximation algorithm for the partitioning problem. It is essential therefore to understand the main ideas and motivations behind this approach. For that reason this algorithm is presented in more detail.

Theorem 3.2 (Feige and Krauthgamer, 2001). *A bisection of cost within an $\mathcal{O}(\log^2 n)$ factor of the minimum can be found in polynomial time.*

Throughout this section the two sides of a (not necessarily optimal) bisection will be denoted as white W and black B. For the analysis, let us fix one of the optimal bisections (arbitrarily) and call it the fixed optimal bisection (W^*, B^*).

3.2.1 Overview

On a high level, the algorithm follows a divide–and–conquer approach. The input graph is recursively divided into parts, using a new cut notion which is called an *amortized cut* (see Definition 3.5), and then the parts are combined into a bisection using dynamic programming.

The algorithm for approximating bisection is based on a subroutine for finding an approximate amortized cut. If the subroutine is guaranteed to find a ρ-amortized cut in a graph, the algorithm computes a bisection whose cost is within ratio of $1 + \mathcal{O}(\rho \log n)$ of the minimum.

In Section 3.2.2 an algorithm for finding a $\rho = \mathcal{O}(\log n)$–amortized cut in a general graph is devised. This yields the desired $\mathcal{O}(\log^2 n)$–approximation factor. The subroutine uses a τ-approximate min-ratio cut in order to find an $\mathcal{O}(\tau)$-amortized cut. The best known approximation algorithms for min-ratio cut in general graphs, due to Leighton and Rao [15, 16] have approximation ratio $\tau = \mathcal{O}(\log n)$.

In certain graph families, there is a better approximation ratio τ for the min-ratio cut problem. If these graph families are closed under taking induced subgraphs, then bisection can be approximated within an improved ratio of $\mathcal{O}(\tau \log n)$. For example, it is shown in [14] that in graphs excluding any fixed graph as a minor (*e.g.* planar graphs) min-ratio cut can be approximated within a constant ratio, *i.e.* $\tau = \mathcal{O}(1)$, thus the minimum bisection can be approximated within a factor $\mathcal{O}(\log n)$.

The algorithm consists of three stages as follows.

1. *Decomposition.* The graph G is recursively divided into parts by a sequence of *divide steps* until it is decomposed into individual vertices. The decomposition can be visualized by a *decomposition tree* T. The root of the tree contains the input graph G, the leaves of the tree contain individual vertices of G, and the two direct descendants of a node i are the two subparts $V_{L(i)}, V_{R(i)}$ obtained in the divide step of its part V_i. The divide step uses an algorithm for finding amortized cuts in a graph (see Section 3.2.2).

2. *Labeling.* The desired outcome of the labeling stage is a labeling of the nodes of T to either black or white which is *α–consistent* with the fixed optimal bisection (W^*, B^*), called in short an *opt–consistent labeling.*

   Definition 3.3. *For a fixed $\frac{1}{2} < \alpha < 1$, the labeling is said to be α–*consistent *with respect to a white–black bisection (W, B) of the input graph if every part V_i satisfies that $|W \cap V_i| \leq \alpha |V_i|$ if the label of node i is white, and that $|B \cap V_i| \leq \alpha |V_i|$ if the label of node i is black.*

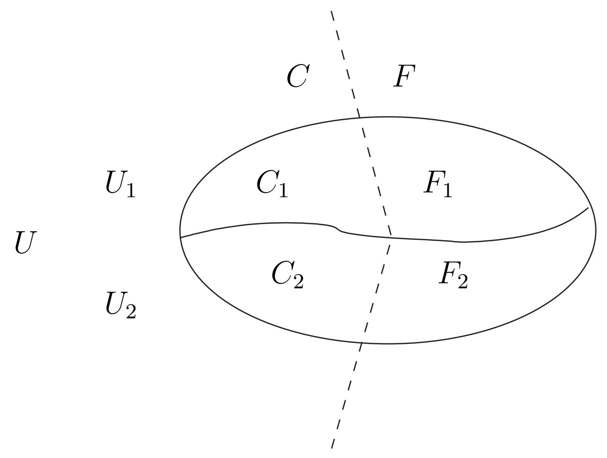

Figure 3.1: A divide step

To be exact, instead of finding an opt-consistent labeling, this stage produces a family of labelings, such that at least one member of the family is opt–consistent.

3. *Combining.* Given a decomposition tree T and a labeling of it, this stage assigns to each vertex v of the graph G a *vertex charge* and the *charge of a bisection* (W, B) of G is defined as the sum of the charge of the vertices. The notion of charge ensures that for every bisection charge is an upper bound on cost. The charge of a vertex—and thus the charge of a bisection as well—depends on the labeling of the decomposition tree. If the charge is defined with respect to an opt–consistent labeling of T then the amortized cuts used in the decomposition stage guarantee that the charge of the fixed optimal bisection (W^*, B^*) is within a polylogarithmic factor of its cost b. Therefore the bisection with *minimum charge* $(\widetilde{W}, \widetilde{B})$ (which can be computed using dynamic programming easily) has cost also within a polylogarithmic ratio of b. To sum up

$$\underline{\underline{\mathcal{O}(\log^2 n) \cdot b}} = \mathcal{O}(\log^2 n) \cdot e(W^*, B^*) \geq charge(W^*, B^*) \geq charge(\widetilde{W}, \widetilde{B}) \geq \underline{\underline{cost(\widetilde{W}, \widetilde{B})}}$$

3.2.2 Decomposition stage

The decomposition stage of the algorithm uses a divide–and–conquer approach. It is desirable that (i) each of the two subproblems can be solved separately; and (ii) the solutions of the two subproblems can be combined while incurring a relatively small additional cost.

Consider a more general cut problem: our aim is to cut off k vertices from the part U. The divide step of part U breaks it into U_1 and U_2, from which k_1 and k_2 vertices have to be cut off, respectively, for certain $k_1 + k_2 = k$. Let us assume that the subproblem associated with each subpart U_i is solved separately (by recursion) and the solution obtained for it is a cut (C_i, F_i) (see also Figure 3.1). The two solution cuts are then combined into a cut $(C, F) := (C_1 \cup C_2, F_1 \cup F_2)$ of U. The cost of the combined cut is given by

$$Cut(U, k) = Cut(U_1, k_1) + Cut(U_2, k_2) + e(C_1, F_2) + e(C_2, F_1) \tag{3.2}$$

where $Cut(X, k)$ denotes the cost of the (solution) cut on X.

However, the formula of the actual cost of the combined cut cannot be used directly in a divide–and–conquer approach, because the cut within each U_i are calculated separately, but the cost depends on both cuts. Therefore the cost of the cut should be estimated. Previous approaches *e.g.* in [7] establish an upper bound on the cost of Equation (3.2) using the total number of edges cut in the divide step

$$Cut(U,k) \leq Cut(U_1,k_1) + Cut(U_2,k_2) + e(U_1,U_2) \tag{3.3}$$

The problem with this approach is that due to the separate calculation of the cuts within each U_i it might happen that only a few edges between the parts U_1 and U_2 end up in the combined cut, yielding a poor upper bound.

The proposed upper bound is the following.

$$Cut(U,k) \leq Cut(U_1,k_1) + Cut(U_2,k_2) + e(C_1,U_2) + e(C_2,U_1) \tag{3.4}$$

The additional term $e(C_1,U_2)$ only depends on the cut in U_1, while $e(C_2,U_1)$ only depends on the cut in U_2, thus completely separating the two problems, allowing a proper divide–and–conquer approach.

We will introduce the notion of charge here, for a more formal definition see Definition 3.6.

Definition 3.4. *The* charge of a divide step *is defined as* $e(C_1,U_2) + e(C_2,U_1)$. *The* charge of a bisection *is the sum of the charges of all the divide steps,* i.e. *applying the bound (3.4) recursively.*

Remark 3.1. *Note that the new accounting method makes a distinction between the two sides C and F of the combined cut. Since we wish to minimize the charge, it makes sense to choose the smaller of the two sides to be C. In our analysis we have a somewhat relaxed condition, requiring that $|C| \leq \alpha|U|$, for a fixed $\frac{1}{2} < \alpha < 1$. The labeling of the decomposition tree corresponds to the identification of side C (see Section 3.2.3).*

We call the vertices of $C = C_1 \cup C_2$ *charged* and the vertices of $F = F_1 \cup F_2$ *free.* The edges in the part U can then be classified as charged–charged, charged–free or free–free, according to their two endpoints.

Instead of finding a bisection of minimum cost, we look for a bisection of minimum charge. Consider the charge of the fixed optimal bisection. The charge of a divide step of a part U is $e(C_1,U_2) + e(C_2,U_1)$ and can be written also as $e(C_1,F_2) + e(C_2,F_1) + 2e(C_1,C_2)$. Observe that a charged–free edge is always an edge of the fixed optimal bisection (and vice versa) and that each edge is cut exactly once in the decomposition stage. So for the fixed optimal bisection, the difference between charge and cost is twice the cost of all the charged–charged edges cut in all the divide steps, therefore the divide step aims at cutting relatively few charged–charged edges. We seek an *amortization scheme* that amortizes the total cost of all charged–charged edges cut against the total cost of all charged–free edges cut (which is exactly b).

Remark 3.2. *The partition of vertices to charged and free is not known to the divide step, we therefore require that the amortization scheme holds for every possible partition of vertices to charged and free.*

The easiest amortization scheme might consider each divide step separately and require that in every divide step the amortized cost is at most ρ, *i.e.* at every part we have that $e(C_1,C_2) \leq \rho[e(C_1,F_2)+e(C_2,F_1)]$. Then the charge of the fixed optimal bisection is clearly at most $(1+2\rho)b$. Unfortunately to respect Remark 3.2 only $\rho = \Omega(n)$ can be guaranteed (see [6] for an example).

New amortization scheme. In the divide step of a part U we amortize $e(C_1,C_2)$ against $e(C,F)$. Since not all the edges in $e(C,F)$ are cut in the divide step, an edge may receive an amortized cost in many divide steps. Our goal is to define the amortization scheme that fulfills the following property.

$$\boxed{\textit{The total cost amortized against a single edge is at most } \mathcal{O}(\rho \log n)\textit{, for a suitable } \rho. \qquad (\dagger)}$$

Corollary 3.1. *It follows that the total cost of the charged–charged edges cut in all the divide steps is at most $\mathcal{O}(\rho \log n) \cdot b$, and so the charge of the fixed optimal bisection is $(1 + \mathcal{O}(\rho \log n)) \cdot b$.*

If each divide step were balanced, the depth of the decomposition tree would be $\mathcal{O}(\log n)$, thus an edge can receive amortized cost in at most $\mathcal{O}(\log n)$ times. Assuming that for each divide step $e(C_1, C_2) \leq \rho \cdot e(C, F)$ holds, then the total cost amortized against a single edge is at most $\mathcal{O}(\rho \log n)$.

However, we do not require that each divide step is balanced, but rather scale the amortization cost according to the imbalance of its divide step. Two different scaling factors are used, and at each step the better will be selected. (W. l. o. g. we assume, that $|U_1| \leq |U_2|$). The first scaling factor is $e(C_1, F_1)/e(C, F)$, and its corresponding amortization method requires that

$$e(C_1, C_2) \leq \rho \cdot \frac{e(C_1, F_1)}{e(C, F)} \cdot e(C, F) = \rho \cdot e(C_1, F_1) \tag{3.5}$$

The second scaling factor is $|C_1|/|C|$, and its corresponding amortization method requires that

$$e(C_1, C_2) \leq \rho \cdot \frac{|C_1|}{|C|} \cdot e(C, F) = \rho \cdot r'(C) \cdot |C_1| \tag{3.6}$$

To prove that property (†) holds for the first amortization method (3.5), one should note that an edge can be inside the smaller side U_1 in at most $\log n$ divide steps. The prove for the second amortization method can be found in [6].

Finally we are able to formally define the desired divide step of the decomposition stage.

Definition 3.5 (Amortized cut). *Let (U_1, U_2) be a cut with $|U_1| \leq |U_2|$ in a graph $G'(U, E')$, and let $U = C \cup F$ be a partition of the vertices to charged vertices C and free vertices F. Let us denote $C_i = U_i \cap C$ and $F_i = U_i \cap F$ for $i = 1, 2$, as in Figure 3.1. Let*

$$\rho_e = \frac{e(C_1, C_2)}{e(C_1, F_1)} \quad \textit{and} \quad \rho_v = \frac{e(C_1, C_2)}{|C_1| \cdot r'(C)} \tag{3.7}$$

We call ρ_e the amortized cost for the edges*, and ρ_v* the amortized cost for the vertices *(note that ρ_e, ρ_v depend on C, F).*

The amortized cost of the cut *(U_1, U_2) is the maximum of $\min\{\rho_e, \rho_v\}$, where the maximum is taken over all partitions $U = C \cup F$ with $0 < |C| \leq \alpha|U|$ for a fixed $\frac{1}{2} < \alpha < 1$. We say that the cut (U_1, U_2) is ρ–*amortized *if its amortized cost is at most ρ.*

Note that in the definition of the amortized cut the condition

$$|C| \leq \alpha|U| \tag{3.8}$$

must hold for the cut. In order to achieve the desired charge value of Corollary 3.1, we need to guarantee this condition in every divide step. The only freedom we have is to select which side will be C and which one will be F in the partition of U. This corresponds to a labeling of the decomposition tree.

First an algorithm for finding a good amortized cut (provided (3.8) holds) will be presented and then the labeling stage in Section 3.2.3 produces a labeling to fulfill (3.8).

Finding a good amortized cut

The aim of this section is to find a $\rho = \mathcal{O}(\log n)$–amortized cut in general graphs. The algorithm is based on finding an approximate min–ratio cut (see Definition 2.5). First we state that an optimal min–ratio cut is an $\mathcal{O}(1)$–amortized cut. As a consequence of that there always exists an $\mathcal{O}(1)$–amortized cut in a graph. Unfortunately, the optimal min–ratio cut is $\mathcal{NP}$–hard to find (Theorem 2.2). Next an algorithm follows which uses a τ–approximate min–ratio cut in order to find an $\mathcal{O}(\tau)$–amortized cut. Best known algorithms (*e.g.* [15]) provide an $\mathcal{O}(\log n)$–approximate min–ratio cut in general graphs and $\mathcal{O}(1)$–approximate ones in graphs excluding any fixed minor [14].

Lemma 3.4. *An optimal min–ratio cut in a graph is $\mathcal{O}(1)$–amortized.* □

Unfortunately, the result of Lemma 3.4 does not extend to an approximate min–ratio cut in straightforward way, but with an additional constraint, as follows.

Lemma 3.5. *Let (V_1, V_2) be a τ–approximate min–ratio cut in a graph, with $|V_1| \leq |V_2|$. If $r(V_1) \leq r(F_1)$ for every $F_1 \subseteq V_1$ then (V_1, V_2) is an $\mathcal{O}(\tau)$–amortized cut.* □

Algorithm 2 Algorithm for finding an $\mathcal{O}(\tau)$–amortized cut

```
1. Find in the input graph $G = (V, E)$ a $\tau$–approximate min–ratio cut $(V_1, V_2)$ with $|V_1| \leq |V_2|$.

2. Create a related graph $G'$
    • Merge all vertices of $V_2$ into a single vertex $t$, removing self loops at $t$, and keeping all
      edges to $V_1$, including parallel edges.
    • Add a new vertex $s$ which is connected to each vertex of $V_1$ by an edge whose capacity
      (weight) is a parameter $p > 0$.

3. Let $S$ denote the vertices of $V_1$ which are on the same side with $s$ in a minimum $(s, t)$–cut
   of $G'$. Find (e.g. by binary search) the minimum $p > 0$ for which $S \neq \emptyset$. (Possibly, $S = V_1$).
   Denote this set by $S^*$.

4. Output the cut $(S^*, V \setminus S^*)$ of the input graph.
```

Lemma 3.6. *The cut $(S^*, V \setminus S^*)$ found by Algorithm 2 is a τ–approximate min–ratio cut. Furthermore $r(S^*) = \min\{r(S') : \emptyset \neq S' \subseteq V_1\}$.*

Proof. It is easy to see that an arbitrary $(S, V \setminus S)$ (s,t)–cut in the related graph has capacity $cap(S) = p|V_1 \setminus S| + e(S, V \setminus S)$. Specially for $S = \emptyset$ this means $cap(\emptyset) = p|V_1|$. Comparing these two values, the empty set yields a smaller value iff $p < \frac{e(S,V\setminus S)}{|S|} = r(S)$. We claim, that the value of p found at Step 3 of Algortihm 2 is $p^* = \min\{r(S) : \emptyset \neq S \subseteq V_1\}$. Indeed, if $p < p^*$, the empty set is better, if $p = p^* + \varepsilon$ for a small positive ε, only a set $S^* \neq \emptyset$ with $r(S^*) = p^*$ will give smaller capacity than the empty set. Therefore $r(S^*) = \min\{r(S') : \emptyset \neq S' \subseteq V_1\}$, as claimed. Furthermore, since $S = V_1$ is also allowed, we get that $r(S^*) \leq r(V_1)$, thus the cut $(S^*, V \setminus S^*)$ is also a τ–approximate min–ratio cut. From Lemma 3.5 it follows that $(S^*, V \setminus S^*)$ is indeed an $\mathcal{O}(\tau)$–amortized cut. □

Theorem 3.3. *Given a subroutine for computing a τ–approximate min-ratio cut, Algorithm 2 finds an $\mathcal{O}(\tau)$–amortized cut.*

Proof. Lemma 3.6 guarantees that the cut found by the algorithm satisfies the requirements of Lemma 3.5, from which it follows that the cut is $\mathcal{O}(\tau)$–amortized. □

3.2.3 Labeling stage

The aim of the labeling stage is to provide an opt–consistent labeling for some fixed $\frac{1}{2} < \alpha < 1$ (recall the Definition 3.3). Unfortunately an opt–consistent labeling is hard to find, so instead this stage produces a family $\mathcal{F}$ of labelings containing at least one opt–consistent one.

The labeling stage first *marks* some of the nodes. The label of these nodes can be arbitrarily chosen (black or white), but it then determines the label of all the other nodes. The marking of T goes from the root of T towards its leaves, as follows. The root of T is always marked, and any other node i in the tree is marked if its closest marked ancestor j satisfies $|V_i| \leq \frac{1}{2\alpha}|V_j|$. Note that $\frac{1}{2} < \frac{1}{2\alpha} < 1$. A labeling is *derived* from the labeling of the marked nodes if the label of an unmarked node is equal to the label of its closed marked ancestor. $\mathcal{F}$ consists the derived labeling of all the possible labelings of the marked nodes. Note that the cardinality of $\mathcal{F}$ can exponential, hence it is not listed explicitly but represented by the marked nodes only.

Lemma 3.7. *$\mathcal{F}$ contains an opt–consistent labeling.*

Proof. Consider a fixed optimal bisection (W^*, B^*). Label the marked nodes in such a way, that each node i receives the label of the color in *minority* among the vertices of V_i and regard the derived labeling ℓ. We claim that ℓ is opt–consistent. The α–consistency condition clearly holds for the marked nodes (since $\alpha > \frac{1}{2}$). The label of an unmarked node i is the same as the label of its closest marked ancestor j. Suppose w. l. o. g. that this label is white. Then $|W^* \cap V_i| \leq |W^* \cap V_j| \leq \frac{1}{2}|V_j| < \frac{1}{2}2\alpha|V_i|$. Hence, ℓ is indeed opt-consistent. $\square$

We can think of a 'good' labeling with respect to a white–black bisection as it labels each node of the tree with the minority color. (Not the exact minority, but α–minority.) To be able to use the amortization scheme of Section 3.2.2 we need the 'smaller' side of each cut to be the charged size. So given a labeling of T we can identify the sets C and F in the following way (see Figure 3.2).

Definition 3.6 (Charge). *Let (W, B) be a bisection of the input graph, and assume we are given a decomposition tree T and a labeling of it. For each (nonleaf) node i of T, if i is labeled white then we let $C_i = W \cap V_i$ and $F_i = B \cap V_i$, and if i is labeled black then we let $C_i = B \cap V_i$ and $F_i = W \cap V_i$. The* charge of the divide step *of a (nonleaf) node i is defined as*

$$e(C_i \cap V_{L(i)}, V_{R(i)}) + e(C_i \cap V_{R(i)}, V_{L(i)})$$

The charge of the bisection (W, B) *is defined as the sum of all the divide steps charges,* i.e.

$$\sum_{i \in T} e(C_i \cap V_{L(i)}, V_{R(i)}) + e(C_i \cap V_{R(i)}, V_{L(i)})$$

Clearly, the charge of the bisection is always an upper bound on its cost, regardless of the labeling. However, if the charge is taken with respect to an α–consistent labeling, it is not much larger than the cost.

Lemma 3.8. *The charge of the bisection (W, B) with respect to an α–consistent labeling is at most $e(W, B) \cdot (1 + \mathcal{O}(\rho \log n))$. Specifically, the charge of the fixed optimal bisection (W^*, B^*) with respect to an opt–consistent labeling is at most $b \cdot (1 + \mathcal{O}(\rho \log n))$.* $\square$

3.2.4 Combining stage

The combining stage finds a bisection $(\widetilde{W}, \widetilde{B})$ of G and a labeling ℓ from $\mathcal{F}$, such that the charge of the bisection with respect to the labeling is minimal over all such bisection–labeling pairs. Lemma 3.7 guarantees that at least one of these labelings is opt–consistent, thus Lemma 3.8 applies. ℓ has minimum charge, which is therefore at most $b \cdot (1 + \mathcal{O}(\rho \log n))$. Using that $\rho = \mathcal{O}(\log n)$ can

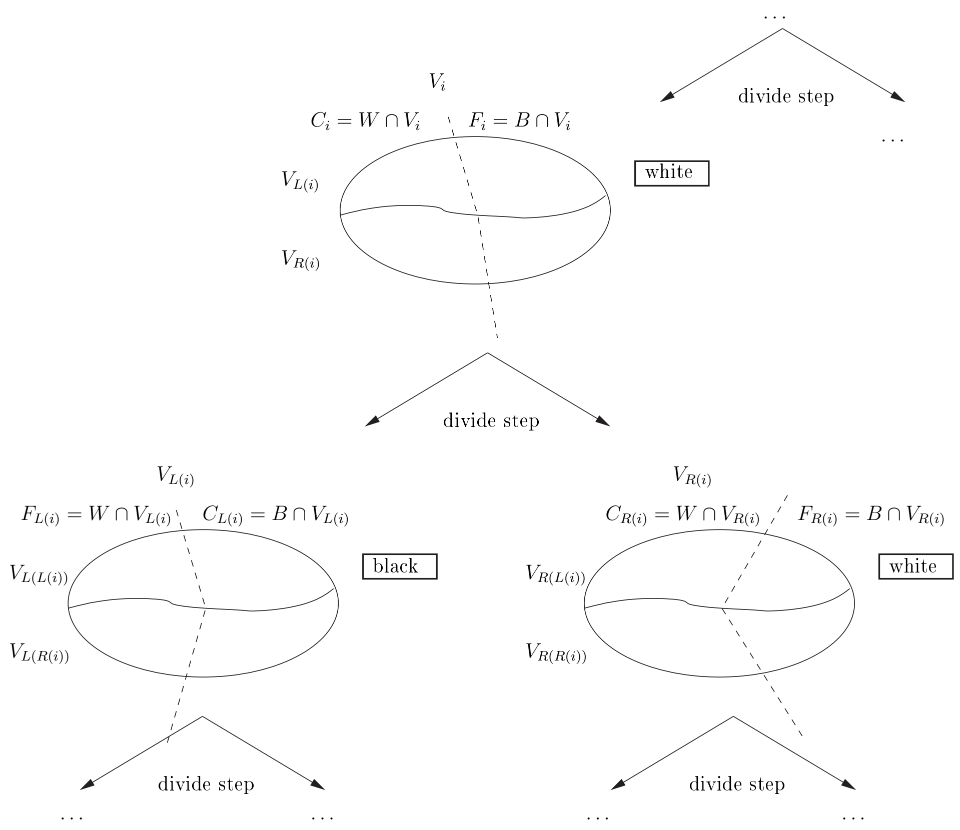

Figure 3.2:
Identifying the charged and free sides of a part in the decomposition tree according to the given labeling

be achieved as was shown in Section 3.2.2 and since charge is an upper bound on cost, the cost of $(\widetilde{W}, \widetilde{B})$ is at most $b \cdot (1 + \mathcal{O}(\log^2 n))$, finishing the proof of Theorem 3.2.

The first observation is that the charge of a bisection can be distributed to the vertices so that the vertex charges will sum up to the bisection charge.

Definition 3.7 (Vertex charge). *For each vertex $v \in V_i$ let the* cross–degree *of v at node i, denoted by $cross_i(v)$, be the cost of the edges that are incident at v and are cut in divide step i. The* charge of a vertex $v \in V$ *is defined as the sum of the cross-degree of v at all nodes i for which v belongs to the charged side,* i.e. $\sum_{i:v \in C_i} cross_i(v)$.

It is easy to see that the charge of a bisection is the sum of the vertex charges. The charge of a vertex depends on (and can be easily computed from) the side of this vertex in the bisection (W, B), but it does not depend on the side of other vertices in the cut. This means that the charge of a bisection depends *linearly* on the placement of the vertices to W or B provided a decomposition tree and a labeling of it is given. On the other hand the cost of a bisection depends quadratically on the placement of the vertices. (The bisection can be expressed as a quadratic integer program, [1]). So charge can be regarded as a linearization of cost within a polylogarithmic approximation factor. It is very important algorithmically, and can be exploited in a dynamic programming scheme. A direct dynamic programming approach—even a pseudo–polynomial one—is not possible because of the quadratic dependence.

Dynamic programming. The dynamic programming table Q has entries of the form $Q(i, k, g)$, where i is a node of the decomposition tree T, k is an integer between 0 and $|V_i|$, and g is a *guess list* that contains the labels of the marked ancestors of node i.

An entry $Q(i, k, g)$ in the table contains the optimal solution to the following problem: Choose k vertices of V_i and a labeling from $\mathcal{F}$ that agrees with g, so that when these k vertices are placed in the side W and the remaining vertices of V_i are placed in the side B, the sum of the charges of all the vertices of V_i with respect to the chosen labeling, is minimal over all such choices.

Clearly, the size of Q is polynomial in n: i and k have $\mathcal{O}(n)$ values, and at each node g may contain the labels of at most $\mathcal{O}(\log n)$ marked ancestors.

Lemma 3.9. *Each cell of Q can be filled out using the table entries of its children in $\mathcal{O}(n)$ time. The whole table can be computed in a bottom–up way in polynomial time.*

Sketch of the proof. For a leaf node i, the table entry $Q(i, k, g)$ can be computed directly, as follows. For the part $V_i = \{v\}$ k can be either 0 or 1. If $k = 0$ then v is necessarily in B, and if $k = 1$ then v is necessarily in W. The guess list g gives the labels of all the nodes on the path from the leaf i to the root, and hence all the labels that can possibly affect the charge of v. So k and g uniquely define all the data that the charge of v depends on, and $Q(i, k, g)$ is just the charge of v. The charge of v is the sum of cross–degrees at ancestor nodes j, where v is in C_j. This happens—recall Definition 3.6 and Figure 3.2—if the label of j according to g agrees with the side of v (which follows from k).

For a nonleaf node i, the table entry $Q(i, k, g)$ can be efficiently computed from table entries of its children nodes $L(i), R(i)$. It is easy to see that

$$Q(i, k, g) = \min_{0 \le j \le k} \min_{g_L, g_R} \{Q(L(i), j, g_L) + Q(R(i), k - j, g_R)\}, \tag{3.9}$$

where g_L and g_R ranges over all possible extensions of g (there are two extensions if the child node is marked and only one, if not). □

The entry $Q(root, \frac{n}{2}, g)$ contains the minimum charge of all bisections of the input graph with respect to any labelings from $\mathcal{F}$, as desired. Since the only ancestor of *root* is itself, g has only two possible values, and $Q(root, \frac{n}{2}, g_i), i = 1, 2$ must be the same due to symmetry.

3.2.5 Generalizations

Previous sections have introduced an $\mathcal{O}(\log^2 n)$–approximation algorithm for the *basic version* of the minimal bisection problem. However, the algorithm can easily be extended to solve other versions of bisection formulations.

1. **Edge–weighted.** The *edge–weighted* version as defined in Definition 2.3 can be solved with the same algorithm but simply using $c(X, Y)$, *i.e.* the cost of edges between to sets $X, Y \subseteq V$, instead of $e(X, Y)$. The corresponding changes in the algorithm and analysis are straightforward. Note that Algorithm 2 for finding the amortized cut requires a subroutine that computes an approximate min-ratio cut with respect to the edge costs, but known algorithms (*e.g.* due to [16]) provide this subroutine. The resulting approximation ratio is the same as for the basic problem, *i.e.* $\mathcal{O}(\log^2 n)$.

2. **Vertex–weighted.** The *vertex–weighted* version as defined in Definition 2.4 can be solved with the same algorithm but rather than considering the number of vertices in a part, we always count its weight. $r(S)$ should now denote $\frac{e(S, V \setminus S)}{\min\{w(S), w(V \setminus S)\}}$. In the dynamic programming phase k ranges from 0 until $w(V_i)$, but it is still polynomial. The corresponding

changes in the algorithm and analysis are straightforward. Note that Algorithm 2 for finding the amortized cut requires a subroutine that computes an approximate min-ratio cut with respect to the vertex weights, but known algorithms (*e.g.* due to [16]) provide this subroutine. The resulting approximation ratio is the same as for the basic problem, *i.e.* $\mathcal{O}(\log^2 n)$.

3. $s-t$ **cut.** In this extension our aim is to find a bisection which separates two special vertices s and t. The dynamic programming table Q should be modified, so that every entry $Q(i,k,g)$ contains two solutions (if they exist); one solution with the k chosen vertices containing s but not t, and vice versa. The corresponding changes in the algorithm and analysis are straightforward. The resulting approximation ratio is the same as for the basic problem, *i.e.* $\mathcal{O}(\log^2 n)$.

4. **Fix number of vertices.** In this extension we wish to cut a fixed number of k vertices from G with minimum edges cut. The dynamic programming stage outputs instead of $Q(root, \frac{n}{2}, g)$ the solution in $Q(root, k, g)$. The corresponding changes in the algorithm and analysis are straightforward. The resulting approximation ratio is the same as for the basic problem, *i.e.* $\mathcal{O}(\log^2 n)$.

Remark 3.3. *Note that most of these extensions can also be combined with each other, for example both edge and vertex costs can be allowed, as the corresponding modifications of the algorithm does not exclude each other.*

3.3 Approximation scheme for dense graphs

In this section a polynomial time approximation scheme (PTAS) will be presented for the minimum bisection problem on everywhere dense graphs. This problem is still $\mathcal{NP}$–hard [4].

Definition 3.8. *A graph is said to be δ–*dense*, if the average degree of the vertices is at least δn (or equivalently it has at least $\delta n^2/2$ edges). It is* everywhere δ–dense *if every vertex has degree at least δn. If $\delta = \Omega(1)$, we will simply use the notion dense and everywhere dense.*

Remark 3.4. *Note that we give a PTAS for minimum bisection on the smaller class of everywhere dense graphs and not on dense graphs. The latter is not easier than a PTAS for minimum bisection on general graphs, as the following reduction shows. Given a general instance with n vertices of the minimum bisection problem, add two disjoint cliques of size $2n$ to it. The resulting graph will be $2/5$–dense, but the value of the minimum bisection remains unchanged.*

The approach relies on a more general theorem of approximating polynomial integer programs (see Theorem 3.4). We need some preparation before this theorem.

Definition 3.9. *A* polynomial integer program (PIP) *is of the form*

$$\begin{array}{lll} \textit{max/min} & p_0(x) & \\ \textit{subject to} & l_i \le p_i(x) \le u_i & \{i = 1, \ldots, m\} \\ & x \in \{0,1\}^n & \end{array}$$

where p_i is a polynomial. When all p_i have degree at most d, the PIP is called degree d PIP.

Since they subsume integer programs, solving PIPs is $\mathcal{NP}$-hard. However, a subclass of PIPs can be approximated in polynomial time.

Definition 3.10. *A degree d polynomial with n variables has* smoothness c *if the value of each coefficient of each degree i term is at most $c \cdot n^{d-i}$.*

*A PIP in which all p_i polynomials are c–smooth with degree at most d is called a c–*smooth degree d PIP.

Definition 3.11. *A solution $a \in \{0,1\}^n$ is said to* satisfy *a constraint $l_i \leq p_i(x) \leq u_i$* within an additive error *δ if $l_i - \delta \leq p_i(a) \leq u_i + \delta$*

Now we are ready to state the general theorem of approximating smooth PIPs. We use the formulation where the objective of the PIP has to be maximized.

Theorem 3.4 (Arora, Karger and Karpinski, 1998). *There is a randomized polynomial–time approximation algorithm that approximately solves smooth PIPs in the following sense. Given a feasible c–smooth degree d PIP with n variables, objective function p_0, the algorithm finds a 0–1 solution z satisfying*

$$p_0(z) \geq OPT - \varepsilon n^d$$

where OPT is the optimum of the PIP. This solution z also satisfies each degree d' constraint within an additive factor of $\varepsilon n^{d'}$ for $d' > 1$, and satisfies each linear constraint within an additive factor of $\mathcal{O}(\varepsilon\sqrt{n \log n})$. The running time of the algorithm is $\mathcal{O}(1/\varepsilon^2)$. The algorithm can be derandomized while increasing the running time by only a polynomial factor.

Proof. The proof can be found in [4]. □

The theorem guarantees a certain *additive* approximation factor. However, it follows that every problem that can be formulated as a smooth PIP and has OPT at least $\mathcal{O}(n^d)$ can be approximated within an ε *multiplicative* factor. Dense graphs have such a property in several optimization problems as was shown in [4]. We now only focus on the bisection problem. Specifically with bisection it is not straightforward to use this theorem and further ideas are needed.

Define to each node v_i a binary variable x_i, which identify the cut. The formulation of the minimum bisection as quadratic PIP is straightforward.

$$\min \quad \sum_{(i,j)\in E} (x_i(1-x_j) + x_j(1-x_i)) \tag{3.10}$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_i = \frac{n}{2} \tag{3.11}$$

$$x_i \in \{0,1\}$$

Note that an edge (i,j) contributes 1 to (3.10) iff $x_i \neq x_j$ and 0 otherwise. The bisection criteria is expressed by (3.11).

Large bisection. If the minimum bisection is large enough, *i.e.* $b \geq \beta n^2$ for a certain β, then Theorem 3.4 gives an assignment z with an objective less than $b + \varepsilon n^2 \leq b(1+\varepsilon/\beta)$, yielding in a multiplicative ε/β–approximation algorithm. The only problem with this is that z fulfills (3.11) only approximately, hence not an exact bisection has been found. But Theorem 3.4 ensures that on both sides of the cut $\frac{n}{2} \pm (\sqrt{n \log n})$ vertices will be found. To balance this cut to a bisection one should move only $\sqrt{n \log n}$ vertices affecting the cut value by at most $\mathcal{O}(n^{1.5} \log n) = o(n^2)$.

Small bisection. If the minimum bisection is less than βn^2 the previous PIP–based approach does not work, therefore a specific algorithm is needed. Algorithm 3 shows the structure of this method. Although this is a random algorithm, it can be easily derandomized.

The main idea behind the algorithm is the so called *exhaustive sampling.* In a good bisection most of the vertices have the majority of their neighbors on their side. So if we knew on which side the majority of neighbors of a vertex v resides, we could place v on this side. Of course the placement of the neighbors is not known. Instead, take a relative small (random) sample set S of vertices, partition it into two sets and judge by this sample. If the great majority of neighbors in

Algorithm 3 Bisection algorithm for δ–everywhere dense graphs with small bisection size

```
1. Pick a set $S$ of $\mathcal{O}(\frac{\log n}{\delta})$ vertices at random.
2. For each possible partition of $S$ into $(S_W, S_B)$ construct a partition $(W, B)$ as follows.
    (a) Let $T$ be the set of vertices that have more than $5/8$ of their neighbors of $S$ in $S_B$. Put
        $T$ in $B$.
    (b) For each vertex $v \notin T$ define $bias(v)$ as
            $\#(\text{neighbors of } v \text{ not in } T) - \#(\text{neighbors of } v \text{ in } T).$
        Put the $\frac{n}{2} - |T|$ vertices with smallest bias into $B$.
3. Output the best bisection found in the previous step.
```

the sample is on one side, then probably the great majority of all the neighbors will be on that side in the optimal bisection, since the graph is everywhere dense. The size of S should be that small that all the possible partitions can be tried—thus the placement according to the optimal bisection (W^*, B^*) is also among them.

It can be proven that with high probability the set T constructed in Step 2a contains every vertex, which 'radically' belongs *e.g.* to the B side, *i.e.* at least $3/4$ of its neighbors are in B^*. Moreover with high probability $T \subset B^*$ and its size is relative large, close to $\frac{n}{2}$. So T should belong to B and it should be extended with a small number of vertices to get an exact bisection. In Step 2b we choose the vertices that are the most strongly connected to T. With the aid of these observations the following theorem can be proved.

Theorem 3.5. *Assuming $b < \beta n^2$, with high probability (over the choice of S in Step 1 of Algorithm 3) the bisection by Algorithm 3 has value at most $b \cdot (1 + \varepsilon)$, where $\varepsilon = 16\beta^2/\delta^2$.* □

3.4 Additive approximation

Finally, we mention the result of Bui and Jones [5], which claims that it is $\mathcal{NP}$–hard to approximate the minimum balanced cut problem within a certain *additive* factor.

Theorem 3.6 (Bui and Jones, 1992). *Unless $\mathcal{P}$=$\mathcal{NP}$ there cannot be a polynomial time algorithm that, given an n–vertex general graph, can find an α–balanced cut with cutsize smaller than $OPT + n^{2-\varepsilon}$, where OPT is the optimum cutsize value, $\varepsilon > 0$ and $\frac{1}{2} \le \alpha < 1$ are fixed constants. The case $\alpha = \frac{1}{2}$ consists of finding the minimum bisection.* □

Note that this theorem does not exclude any multiplicative approximation factor ρ. To see this, let us fix an $\varepsilon > 0$ in the additive term of Theorem 3.6 and $\alpha = \frac{1}{2}$. In order not to confront with the theorem, for a ρ–approximation algorithm it must hold that

$$b(1 + \rho) \ge b + n^{2-\varepsilon},$$

which implies the following on ρ

$$\rho \ge \frac{n^{2-\varepsilon}}{b}.$$

If $b = \mathcal{O}(n^2)$ (which is easily possible) then

$$\frac{n^{2-\varepsilon}}{b} \overset{n\to\infty}{\longrightarrow} 0$$

thus ρ can be arbitrary small.

There is no theoretical hardness evidence that a PTAS to the general minimal bisection problem cannot exist, although the best known approximation algorithm is the one presented in Section 3.2 and has 'only' polylogarithmic approximation ratio. It remains an open problem to close this gap.

Chapter 4

Approximation of the partitioning problem

In this chapter we will use the techniques for approximating the minimum bisection presented in the previous chapter to develop approximation algorithms for (some versions of) the PART3 partitioning problem (recall Definition 2.8).

At first glance the two problems seem to be rather different.

1. The symmetry of the two sides of the partition is broken: we now distinguish between hardware and software side.
2. Instead of just minimizing the weight of crossing edges, we aim at minimizing the weight of crossing edges plus the weight of the software side.
3. Only the size of one side of the cut is constrained and the constraint is not strict, but an upper bound is given.

We should go through these differences and reason why the bisection problem and especially the algorithm of Feige and Krauthgamer (hereinafter F&K) despite the differences is a good candidate to consider in developing approximation algorithms for the partitioning problem.

Concerning the first difference, in the approach of Section 3.2 the two sides of the bisection are indeed distinguished (see Remark 3.1)—although it is not in the nature of the bisection problem.

Remember how the cost of the sub–solutions were assembled in the divide step of the F&K bisection algorithm to result in an overall solution. As the additional cost of software weights can be minimized independently in the subproblems, a similar decomposition approach could be beneficial in the partitioning problem as well, neutralizing the second difference.

As for the third difference, the dynamic programming stage of F&K is general enough to be used for other purposes as well. It computes the optimal solution for several subproblems that could be assembled to the optimum of other problem formulations. On the other hand, a reduction of the partitioning problem to the bisection by balancing the hardware and software sides with additional vertices also seems to be possible.

This chapter is organized as follows. Based on the algorithm of F&K, we achieve the same approximation factor for the PART3 problem in Section 4.1 in two different ways: first, we *reduce* the PART3 problem to an extension of the minimum bisection (as outlined in Section 3.2.5) in Section 4.1.1. Second, we *adapt* the algorithm of F&K to the PART3 problem in Section 4.1.2.

4.1 Polylogarithmic approximation

Since basically we want to use a bisection algorithm for our PART3 problem, there is no hope to handle arbitrary vertex weights, but polynomial vertex weights only (remember Remark 2.1). Note that PART3 is $\mathcal{NP}$–hard in the strong sense as well (see Theorem 2.6). Our aim is now to prove the following theorem in two different ways.

Theorem 4.1. *The* PART3 *problem with polynomial integer hardware weights, and arbitrary non–negative software and communication weights can be approximated within an* $\mathcal{O}(\log^2 n)$ *factor.*

4.1.1 Reduction

Recall the definition of the PART3 problem. Given a graph $G(V,E)$ of n vertices with two vertex costs $s,h : V \to \mathbb{R}^+$ and edge costs $c : E \to \mathbb{R}^+$, furthermore a hardware limit H_0 our aim is to solve the following (not linear) program.

$$\min \quad s(V_S) + c(V_H, V_S) \tag{4.1a}$$

$$\text{subject to} \quad h(V_H) \le H_0 \tag{4.1b}$$

over all (HW–SW) partitions of G, $V = V_H \uplus V_S$.

Our aim is to prove Theorem 4.1 through a reduction of (a relaxed version of) the PART3 problem to the minimal bisection problem.

In the first step of the reduction we define an (almost) equivalent PART3 problem instance with software costs everywhere zero. Let $G'(V',E')$ defined as follows (see Figure 4.1).

$$V' := V \cup \{x\} \text{ and } E' := E \cup \{(v,x) : \forall v \in V\}.$$

and let $H_0' := H_0$. The cost functions s', h' and c' are also modified.

$$s' :\equiv 0,$$

while

$$h'(v) := \begin{cases} h(v), & \forall v \in V \\ 0, & \text{if } v = x \end{cases}$$

and

$$c'(u,v) := \begin{cases} c(u,v), & \forall u,v \in V \\ s(u), & \text{if } v = x \end{cases}$$

Proposition 4.1. *The* PART3 *problem as defined in* (4.1a)–(4.1b) *is equivalent with the following optimization problem.*

$$min_{x\in V_H'} \quad c'(V_H', V_S') \tag{4.2a}$$

$$\textit{subject to} \quad h'(V_H') \le H_0' \tag{4.2b}$$

over all (HW–SW) partitions of G', $V' = V_H' \uplus V_S'$.

Proof. Consider a partition (V_H', V_S') of G'. It also induces a cut (V_H, V_S) on G with $V_H = V_H' \setminus \{x\}$ and $V_S = V_S'$. Since the new node x is fixed to hardware, the cost of this cut is $c(V_H', V_S') = c(V_H, V_S) + c'(x, V_S) = c(V_H, V_S) + s(V_S)$, as desired. The set of feasible solutions is obviously the same, as the hardware cost of x is defined as zero. $\square$

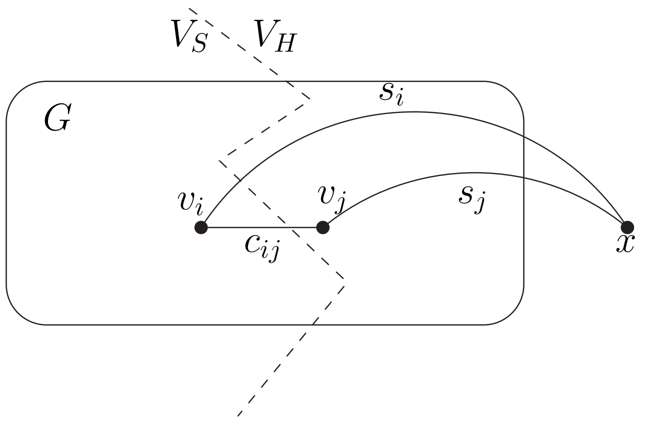

Figure 4.1: The reduced problem instance

It follows from Proposition 4.1 that it is enough to deal with the (4.2a)–(4.2b) formulation, that is to solve the PART3 problem without software costs present and with the small additional constraint that one special vertex should be fixed to hardware.

We have restricted the general PART3 problem to the class with polynomial hardware costs, *i.e.* $h(v_i) \leq n^t, \forall v_i \in V$ for an appropriate constant t. It also implies that H_0 should also be polynomial in n, since $H_0 \geq h(V)$ would be a meaningless constraint. As a consequence of this we have enough time to exhaustively search for the best hardware cost value H^* and solve at most H_0 instances of the following problem.

$$\min_{x \in V_H} \quad c(V_H, V_S) \tag{4.3a}$$

$$\text{subject to} \quad h(V_H) = H \tag{4.3b}$$

over all (HW–SW) partitions of G, $V = V_H \uplus V_S$.

Our aim is now to reduce this problem to the edge– and vertex–weighted minimal bisection problem. Now let us add two special vertices y_h and y_s to the graph with vertex weight $h(y_h) := h(V)$ and $h(y_s) := 2H$, and find a bisection in this graph. The total vertex weight in the resulting graph is $h(V \cup \{y_h, y_s\}) = 2h(V) + 2H$, thus a bisection of the extended graph has vertex weight $w := h(V) + H$ on each side. Since $h(y_h) + h(y_s) = h(V) + 2H > w$, y_h and y_s are on different sides of the bisection. Such a bisection induces a cut (V_H, V_S) in G: the side containing y_h has vertex weight $w - h(y_h) = H$ in G and the other has vertex weight $w - h(y_s) = h(V) - H$ in G. The side of the bisection containing y_h will be identified as hardware, thus it should contain x as well. So it must further be guaranteed, that x and y_h belong to the same side. To achieve this, add a new edge (x, y) with cost $c(x, y) := \infty$ to the graph—a very large edge cost *e.g.* $c(E)$ is sufficient. This results in the following proposition.

Proposition 4.2. *The problem of* (4.3a)–(4.3b) *is equivalent with the following optimization problem. Find the vertex–weighted bisection with minimum edge weight in $G''(V'', E'')$, where $V'' := G \cup \{y_h, y_s\}$, $E'' := E \cup \{(x, y)\}$, $h(y_h) := h(V)$, $h(y_s) := 2H$ and $c(x, y) := c(E)$.* □

The resulting problem is an edge– and (polynomially bounded) vertex–weighted minimum bisection problem in a graph. This—according to extension 1 and 2 of Section 3.2.5 and Remark 3.3—can be approximated within an $\mathcal{O}(\log^2 n)$ factor, yielding the same approximation for our initial problem, that is the PART3 problem with polynomial hardware weights. This completes the proof of Theorem 4.1.

Remark 4.1. *If $H \leq \frac{h(V)}{2}$, then it is enough to add one vertex y_h with vertex weight $h(y_h) := h(V) - 2H \geq 0$ to the graph in the last step of the reduction, and find the minimum bisection in this graph. If $H > \frac{h(V)}{2}$ then this construction would yield an undesired negative vertex weight.*

Remark 4.2. *Instead of solving the problem* (4.2a)–(4.2b) *we solve H_0 instances of the problem* (4.3a)–(4.3b)*, which is not a very elegant solution. An obvious way to attack* (4.2a)–(4.2b) *directly would be to establish a similar reduction to the (weighted) minimal α–balanced cut problem, as follows.*

We again add two vertices y_h, y_s to the graph with appropriate weights w_h and w_s. Denoting by $W := h(V(G'')) = h(V) + w_h + w_s$ the total (hardware) weight of the extended graph, the reduction works if w_h, w_s fulfill the following properties.

$$w_h + w_s > \alpha W \tag{4.4}$$
$$w_h + H_0 = \alpha W \tag{4.5}$$
$$w_h \geq (1-\alpha)W \tag{4.6}$$

(4.4) guarantees that w_h and w_s are on different sides of an α–balanced cut of G'', while (4.5) and (4.6) ensure that an α–balanced cut of G'' will cut off at most H_0 vertex weight of G and every cut with at most H_0 vertex weight in G corresponds to an α–balanced cut of G'' when extended with y_h. It is easy to see that appropriate w_h, w_s exist for any $\frac{1}{2} \leq \alpha < 1$ and they are polynomial in n. As before it should be ensured with an additional edge that both y_h and x reside on the hardware side, completing the reduction.

Unfortunately to our best knowledge there is no approximation algorithm known to the vertex–weighted α–balanced cut problem. Any such approximation factor would immediately yield the same factor for the PART3 problem.

There are so–called pseudo–approximation algorithms for the α–balanced cut problem. This means that the algorithm guarantees a certain approximation factor not with respect to the optimum α–balanced cut, but with respect to the optimum α'–balanced cut for some $\alpha' > \alpha$. For instance the following theorem holds.

Theorem 4.2. *There is a polynomial time algorithm that finds an α–balanced cut for $\frac{1}{2} < \alpha < 1$ with cost at most $\mathcal{O}(b \log n)$, where b is the cost of the optimal bisection.* □

Note that the case $\alpha = \frac{1}{2}$ is excluded in the above theorem. However, such a theorem does not imply any *real* approximation ratio for the balanced cut problem, which justifies our original approach of reducing to the bisection problem instead of the α–balanced cut problem.

4.1.2 Modification of the algorithm

In this section we follow the approach of F&K and adapt their algorithm to the PART3 problem by slightly modifying its steps and definitions. Our aim is again to guarantee a polylogarithmic approximation ratio as stated in Theorem 4.1. Instead of their basic algorithm we use the vertex–weighted version as reference. The hardware costs in PART3 play the role of the vertex weights. For the sake of comparability we will indicate a HW–SW partition with (W, B), where B means the software side.

In the decomposition stage the problem was recursively cut into subproblems and the solution of the subproblems were assembled to an overall solution. The cost of the assembled solution was estimated using the cost of the subproblems as given in (3.4). Fortunately the same estimation can be used for the partitioning problem as well, since the additive cost factor of the software costs sum up directly for the subproblems—and thus it should not be estimated at all. If $Cut(U, H)$ denote the cost of the partitioning problem on part U with hardware limit H, the following inequality holds.

$$Cut(U, H_0) = Cut(U_1, H_1) + Cut(U_2, H_2) + c(C_1, U_2) + c(C_2, U_1) \tag{4.7}$$

for a certain $H_1 + H_2 = H_0$. Our goal is to do the decomposition stage in exactly the same way as before. Therefore the notion of charge (as the upper bound on cost) should be redefined to handle the modified cost metric: it should incorporate the software cost of the vertices as well. So the new charge should contain both the sum of the divide steps as implied by (4.7) and the software costs of the vertices put into software. The following is a modification of Definition 3.6.

Definition 4.1 (Modified charge). *Let (W, B) be a HW–SW partitioning of the input graph, and assume we are given a decomposition tree T and a labeling of it. For each (nonleaf) node i of T, if i is labeled white then we let $C_i = W \cap V_i$ and $F_i = B \cap V_i$, and if i is labeled black then we let $C_i = B \cap V_i$ and $F_i = W \cap V_i$. The* charge of the divide step *of a (nonleaf) node i is defined as*

$$c(C_i \cap V_{L(i)}, V_{R(i)}) + c(C_i \cap V_{R(i)}, V_{L(i)})$$

The charge of the HW–SW partition (W, B) *is defined as the sum of all the divide steps charges and the software costs of the vertices in B,* i.e.

$$s(B) + \sum_{i \in T} c(C_i \cap V_{L(i)}, V_{R(i)}) + c(C_i \cap V_{R(i)}, V_{L(i)})$$

It might be strange at first that now we define the charge with respect to a HW–SW partition instead of a bisection. Note that the approach of F&K serves more general purposes. As was shown in Section 3.2.5 it can handle the problem of cutting away an arbitrary k number of vertices without modifying the algorithm. So the propositions of the algorithm remain true even if (W, B) denotes any $(k, n-k)$ partition of the graph.

So our proposed modification of the charge notion does not affect its properties. It is still an upper bound of cost and a similar statement as Lemma 3.8 can be claimed, *i.e.* for an optimal HW–SW partition the modified charge—if defined with respect to an opt–consistent labeling—will be within a logarithmic factor of its cost (since the new additive factor in the charge equals the new additive factor in the cost). The definition of an amortized cut remains the same and the rest of the decomposition and the labeling stage is unmodified.

However, the combining stage has to be modified according to the modified charge notion. Remember, that the aim of the combining stage was to find the best bisection–labeling pair that minimizes charge provided a decomposition tree and a family $\mathcal{F}$ of labelings is given. The dynamic programming approach relied on the fact that the charge can be distributed among the vertices so that the vertex charges (recall Definition 3.7) sum up to the overall charge. We should modify the definition of vertex charge by adding the software cost to the charge of vertices put in B.

Definition 4.2 (Modified vertex charge). *For each vertex $v \in V_i$ let the* cross-degree *of v at node i, denoted by $cross_i(v)$, be the cost of the edges that are incident at v and are cut in divide step i. The* charge of a vertex $v \in V$ *is defined as the sum of the cross-degree of v at all nodes i for which v belongs to the charged side plus the software cost of the vertex if it belongs to B,* i.e. *$s(v)\chi_B(v) + \sum_{i:v \in C_i} cross_i(v)$, where χ_B denotes the characteristic function of the set B.*

At first glance the condition whether v is in B or not might be surprising in the definition, but note that the vertex charge depends on the side of the vertex in the bisection in the original definition as well, as the cross degrees are summed up only if the vertex is in the charged size.

The properties of vertex charge remain the same after this modification. Clearly the vertex charges sum up to the charge of the bisection as before. The charge of a vertex further on depends only on its side and is independent of the side of other vertices.

Therefore the dynamic programming stage of the algorithm mainly remains the same and a similar statement as of Lemma 3.9 can be proved (recall this proof). The definition of $Q(i, k, g)$ is the same, but k goes now only until $\max\{H_0, h(V_i)\}$, since we are only interested in partitions satisfying the hardware constraint. The recursion of Equation (3.9) can further on be used. However, the initial step of the dynamic programming should slightly be modified according to the new vertex charge notion: when i is a leaf node, $V_i = \{v\}$, and $k = 0$, *i.e.* $v \in B$, then $s(v)$ should be added to the (old) vertex charge value, thus also to $Q(i, k, g)$.

With this modification, the entries $Q(root, k, g)$ will contain the minimal (new) charge value for cutting the graph into two parts of size k and $n - k$. According to the definition of Q this side is the the hardware side W, and since $k \leq H_0$, these are feasible solutions for the PART3 problem. The output of the algorithm is the smallest value among $Q(root, k, g)$, where $0 < k \leq H_0$. As our modified charge still has the property of approximating the cost, the output HW–SW partitioning is again within a polylogarithmic factor of the optimum, completing the proof of Theorem 4.1.

Chapter 5

Conclusions

This work presents a novel hardware–software partitioning model and several problem formulations on this model. First the hardness of the defined partitioning problems has been clarified. It turned out that some versions are polynomially solvable, while others are $\mathcal{NP}$–hard. The main goal of the thesis was to develop approximation algorithms for the hard versions of the hardware–software partitioning problem.

Many approximation algorithms for graph problems use the divide–and–conquer approach to break the original problem into loosely coupled smaller ones that are easier to solve. This motivates the search for balanced cuts or bisections.

In the first part of this thesis we give an overview of existing bisection approximation algorithms, which are of independent interest. In the second part we use the approach of Feige and Krauthgamer, that gives the best known approximation ratio currently to the minimal bisection problem, to develop approximation algorithms to the hardware–software partitioning problem.

The partitioning model, the partitioning problem formulations, the $\mathcal{NP}$–hardness proofs and the two derived approximation algorithms are my own results.

Acknowledgements

I would like to thank to my supervisor András Benczúr and to my colleague Zoltán Mann for their useful comments, which helped improve the quality of this work significantly.

Bibliography

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

[2] P. Arató, Z. Á. Mann, and A. Orbán. Algorithmic aspects of hardware/software partitioning. Submitted to ACM TODAES.

[3] P. Arató, Z. Á. Mann, and A. Orbán. Hardware-software co-design for Kohonen's self-organizing map. In *Proceedings of the IEEE 7th International Conference on Intelligent Engineering Systems*, 2003.

[4] Sanjeev Arora, David Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. *Journal of Computer and System Sciences (JCSS)*, 58(1):193–210, 1999.

[5] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is np-hard. *Information Processing Letters*, 42:153–159, May 1992.

[6] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Journal of Computing*, 31(4):1090–1118, 2002.

[7] U. Feige, R. Krauthgamer, and N. Nissim. Approximating the minimum bisection size. In *Symposium on the Theory of Computing (STOC)*, pages 530–536, 2000.

[8] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[9] M. R. Garey and D. S. Johnson. *A guide to the theory of NP–completeness.* Freeman, San Francisco, 1979.

[10] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):216–227, 1980.

[11] Naveen Garg, Huzur Saran, and Vijay V. Vazirani. Finding separator cuts in planar graphs within twice the optimal. *SIAM Journal of Computing*, 29(1):159–179, 1999.

[12] R. Gomory and T. C. Hu. Multi–terminal netwotk flows. *SIAM Journal of Applied Mathematic*, 9:551–570, 1961.

[13] D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing, Boston, MA, 1997.

[14] Philip N. Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Symposium on the Theory of Computing (STOC)*, pages 682–690, 1993.

[15] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *IEEE Symposium on Foundations of Computer Science*, pages 422–431, 1988.

[16] 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.

[17] M. Lopez-Vallejo and J. C. Lopez. On the hardware-software partitioning problem: system modeling and partitioning techniques. *ACM Transactions on Design Automation of Electronic Systems*, 8(3):269–297, July 2003.

[18] Z. Á. Mann and A. Orbán. Optimization problems in system-level synthesis. In *Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, 2003.

[19] C. H. Papadimitriou. *Computational complexity.* Addison Wesley, 1994.

[20] Huzur Saran and Vijay V. Vazirani. Finding the minimum cut within twice the optimum. *SIAM Journal of Computing*, 24(1):101–108, 1995.

[21] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37(2):318–334, April 1990.

[22] H. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, 3(1):85–93, Jan 1977.