Partitioning a Square into Rectangles: NP-Completeness and Approximation Algorithms

Olivier Beaumont, Vincent Boudet, Fabrice Rastello, Yves Robert

Laboratoire de l'Informatique du Parallélisme

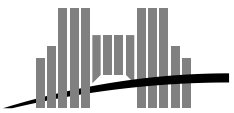
École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

Partitioning a Square into Rectangles: NP-Completeness and Approximation Algorithms

Olivier Beaumont
Vincent Boudet
Fabrice Rastello
Yves Robert

February 2000

Research Report N° 2000-10



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

Partitioning a Square into Rectangles: NP-Completeness and Approximation Algorithms

Olivier Beaumont
Vincent Boudet
Fabrice Rastello
Yves Robert

February 2000

Abstract

In this paper, we deal with two geometric problems arising from heterogeneous parallel computing: how to partition the unit square into p rectangles of given area $s_1, s_2, \ldots, s_p$ (such that $\sum_{i=1}^{p} s_i = 1$), so as to minimize (i) either the sum of the p perimeters of the rectangles (ii) or the largest perimeter of the p rectangles. For both problems, we prove NP-completeness and we introduce approximation algorithms.

Keywords: heterogeneous resources, load-balancing, communication cost, parallel computing, partitioning, NP-completeness, geometric problems

Résumé

Dans ce rapport, nous nous intéressons à deux problèmes géométriques issus de calculs parallèles hétérogènes : comment découper le carré unité en p rectangles d'aires donnés $s_1, s_2, \ldots, s_p$ (tel que $\sum_{i=1}^{p} s_i = 1$), de manière à minimiser (i) soit la somme des périmètres des p rectangles (ii) soit le plus grand périmètre de ces p rectangles. Pour les deux problèmes, nous établissons leur NP-complétude et nous introduisons des algorithmes d'approximation.

Mots-clés: ressources hétérogènes, équilibrage de charge, cout de communication, calcul parallèle, découpage, NP-complétude, problèmes géométriques

1 Introduction

In this paper, we deal with two simple geometric problems: how to partition the unit square into p rectangles of given area $s_1, s_2, \ldots, s_p$ (such that $\sum_{i=1}^{p} s_i = 1$), so as to minimize

- either the sum of the p half perimeters of the rectangles,
- or the largest half perimeter of the p rectangles.

Note that there always exist solutions to these problems: e.g. tile the unit square into p horizontal slices of height $s_1, s_2, \ldots, s_p$. The difficulty is to minimize the objective function.

Consider the following example with $p = 5$ rectangles $R_1, \ldots, R_5$ of areas $s_1 = 0.36$, $s_2 = 0.25$, $s_3 = s_4 = s_5 = 0.13$. A possible partition is shown in Figure 1. The size of each rectangle is the following: $0.61 \times \frac{36}{61}$ for R_1, $0.61 \times \frac{25}{61}$ for R_2, and $0.39 \times \frac{1}{3}$ for R_3, R_4, and R_5. The maximum half-perimeter is that of R_1, approximately 1.2002, which is very close to the absolute lower bound 1.2 obtained when the largest rectangle is a square (this is not achievable in this example). As for the second objective function, we compute that the sum of the half-perimeters is 4.39, while the absolute lower bound is $\sum_{i=1}^{p} 2\sqrt{s_i} \approx 4.36$ (obtained when all rectangles are squares, which is not achievable in this example either). Hence the partition turns out to be very satisfactory for both objective functions. The geometric interpretation for the sum of the half-perimeters is nice: it is the length of the lines drawn to make the partition, plus 2 corresponding to the right and bottom edge of the unit square.

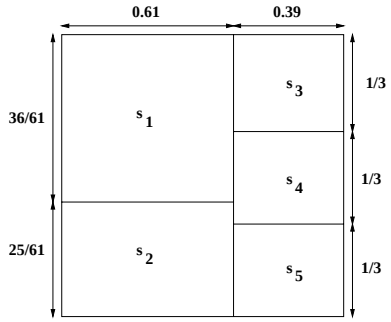


Figure 1: A simple example with 5 rectangles.

The main results of the paper are the proof of NP-completeness and approximation algorithms for both optimization problems. Beforehand, we explain the initial motivation for this work, which arises from minimizing communications in the design of parallel algorithms targeted to heterogeneous platforms. The rest of the paper is organized as follows. In Section 2 we explain the motivation from heterogeneous parallel computing. In Section 3 we formally state the optimization problems PERI-SUM (minimize the sum of the perimeters of the rectangles) and PERI-MAX (minimize the largest perimeter), and we establish their NP-completeness. Section 4 is devoted to the design of approximation algorithms for PERI-MAX; Section 5 is its counterpart for PERI-MAX. In Section 6 we briefly survey related optimization problems. We give some final remarks and conclusions in Section 7.

2 Problem Motivation

The motivation for this work is the design of parallel Matrix-Matrix Multiplication (MMM for short) algorithms targeted to heterogeneous platforms, such as heterogeneous clusters of workstations, or

collections of such clusters.

Parallel MMM algorithms work as follows: let $C = A \times B$ the product to be computed, where A and B are square matrices of size $n \times n$. First, granularity is increased: matrix blocks rather than elementary matrix coefficients are allocated to processors, as in the ScaLAPACK library [4]. Hence, each "element" in A, B and C is a square $r \times r$ block, and the unit of computation is the updating of one block, i.e. a matrix-matrix multiplication of size r. Assume there are p processors. The three matrices A, B and C are partitioned into p (superposed) rectangles. There is a one-to-one mapping between these rectangles and the processors. Each processor is responsible for updating its rectangle: at each step, one pivot column and one pivot row are communicated to all processors, and independent updates take place; more precisely, each processor updates each block in its rectangle with one block from the pivot row and one block from the column row, as illustrated in Figure 2.

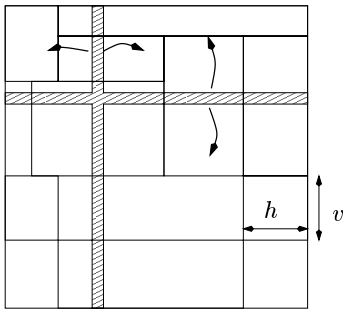


Figure 2: The MMM algorithm on a heterogeneous platform.

Using different-speed processors, we want to balance the computing load so that each processor receives an amount of work in accordance to its computing power. Because all C blocks require the same amount of arithmetic operations, each processor executes an amount of work which is proportional to the number of blocks that are allocated to it, hence proportional to the area of its rectangle. In Figure 2, we have 13 different-speed computing resources. We let s_i the fraction of the total computing power represented by processor P_i, $1 \leq i \leq p$. Normalizing processor speeds, we have $\sum_{i=1}^{p} s_i = 1$. Normalizing the computing workload accordingly, we have to tile the unit square into p rectangles R_i of prescribed area s_i, $1 \leq i \leq p$. The question is: how to compute the *shape* of these p rectangles so as to minimize the total execution time?

Let $h_i \times v_i$ be the size of rectangle R_i, where $h_i v_i = s_i$. At each step of the MMM algorithm, communications take place between processors: the total volume of data exchanged is proportional to the sum $\hat{C} = \sum_{i=1}^{p}(h_i + v_i)$ of the half perimeters of the p rectangles R_i. In fact, this is not exactly true: because the pivot row and columns are not sent to the processors that own them, we should subtract 2 from $\hat{C}$, 1 for the horizontal communications and 1 for the vertical ones. Since minimizing $\hat{C}$ or $\hat{C} - 2$ is equivalent, we keep the value of $\hat{C}$ as stated. Minimizing $\hat{C}$ seems to be a very natural goal, because it represents the total volume of communications. For instance it is natural to assume that communications will be mostly sequential on a heterogeneous network of workstations where processors are linked by a simple Ethernet network; also, there will be little or none computation/communication overlap on such a platform. In that context, minimizing the total communication volume is the main objective.

Conversely, some communications can occur in parallel, if the computing resources are linked through a dedicated high-speed network, and if parallel communication links are provided. In that context, we may want to minimize the maximal amount of communications to be performed by each processor, so that the objective function becomes $\hat{M} = \max_{1 \leq i \leq p}(h_i + v_i)$.

Once a solution to either optimization problem has been found, we derive the allocation of data elements to processor P_i by rounding up the values $n \times h_i$ and $n \times v_i$. Finally, note that both optimization problems have a wide potential applicability. Forgetting about MMM algorithms, consider the implementation of any application (such as a finite-difference scheme) where heterogeneous processors communicate boundary elements at each step (the communication scheme need not be nearest-neighbor, it can be anything): minimizing the total communication volume, or the maximal amount of communications performed by one processor, while load-balancing the work, amounts to solving exactly the same optimization problems.

3 NP-Completeness

3.1 Problem Formulation

We formally state both optimization problems. We have to determine p rectangles R_i, of prescribed area s_i, $1 \leq i \leq p$ where $\sum_{i=1}^{p} s_i = 1$. The shape of each R_i is the degree of freedom: we want to tile the unit square so as to solve the following optimization problems:

Definition 1 *PERI-SUM(s): Given p real positive numbers $s_1, \ldots, s_p$ s.t. $\sum_{i=1}^{p} s_i = 1$, find a partition of the unit square into p rectangles R_i of area s_i and of size $h_i \times v_i$, so that $\hat{C} = \sum_{i=1}^{p}(h_i + v_i)$ is minimized.*

Definition 2 *PERI-MAX(s): Given p real positive numbers $s_1, \ldots, s_p$ s.t. $\sum_{i=1}^{p} s_i = 1$, find a partition of the unit square into p rectangles R_i of area s_i and of size $h_i \times v_i$, so that $\hat{M} = \max_{1 \leq i \leq p}(h_i + v_i)$ is minimized.*

There is an obvious lower bound for PERI-SUM(s) and for PERI-MAX(s):

Lemma 1 *For all solutions of PERI-SUM(s), $\hat{C} \geq 2\sum_{i=1}^{p} \sqrt{s_i}$. For all solutions of PERI-MAX(s), $\hat{M} \geq 2 \max_{1 \leq i \leq p} \sqrt{s_i}$.*

Proof The half-perimeter of each rectangle R_i will be always larger that $2\sqrt{s_i}$, the value when it is a square. Of course, tiling the unit square into p squares of area s_i is not always possible, so the lower bound for PERI-SUM(s) is not always tight. The same observation holds for PERI-MAX(s), as shown by the example in Section 1. ■

3.2 PERI-SUM(s)

The decision problem associated to the optimization problem PERI-SUM is the following:

Definition 3 *PERI-SUM(s,K): Given p real positive numbers $s_1, \ldots, s_p$ s.t. $\sum_{i=1}^{p} s_i = 1$ and a positive real bound K, is there a partition of the unit square into p rectangles R_i of area s_i and of size $h_i \times v_i$, so that $\sum_{i=1}^{p}(h_i + v_i) \leq K$?*

Our first result states the intrinsic difficulty of the PERI-SUM optimization problem:

Theorem 1 *PERI-SUM(s,K) is NP-complete.*

Proof We give the main ideas of the proof. A full-length version is available in the technical report [3]. Obviously, PERI-SUM(s,K) $\in$ NP. We use the following reduction:

Lemma 2

$$\textit{2P-eq} \leq_P \textit{SSP} \leq_P \textit{PERI-SUM},$$

where SSP and 2P-eq are defined as follows:

Definition 4 *2-Partition-Equal (2P-eq)*
Given a set of p integers $\mathcal{A} = \{a_1, \ldots, a_p\}$, is there a partition of $\{1, \ldots, p\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that

$$\sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i \textit{ and } card(\mathcal{A}_1) = card(\mathcal{A}_2) \quad ?$$

Definition 5 *Square-Square-Partition (SSP)*
Given a set $\mathcal{A} = \{s_1, \ldots, s_p\}$ of p real positive numbers such that $\sum_{i=1}^{p} s_i = 1$, is there a partition of the unit square into p squares S_i of area s_i ?

Since 2P-eq is known to be NP-Complete [6], Lemma 2 will complete the proof of Theorem 1.

3.2.1 Reduction: SSP $\leq_P$ PERI-SUM(s,K)

We start by proving the easy part of Lemma 2, i.e. SSP $\leq_P$ PERI-SUM(s,K). Let $\mathcal{A} = \{s_1, \ldots, s_p\}$ be a set of p real positive numbers s.t. $\sum_{i=1}^{p} s_i = 1$. Solving SSP is equivalent to solving PERI-SUM(s,K) with

$$K = 2 \sum_{i=1}^{p} \sqrt{s_i}$$

and therefore,

$$\text{SSP} \leq_P \text{PERI-SUM}.$$

3.2.2 Reduction: 2P-eq $\leq_P$ SSP

In this section, we consider an arbitrary instance of the 2-Partition-Equal problem, i.e. a set $\mathcal{A} = \{a_1, \ldots, a_n\}$ of n integers. We assume that $n > 400$ without loss of generality. We have to polynomially transform this instance into an instance of the SSP problem which has a solution iff the original instance of 2-Partition-Equal has one solution.

Define $\{b_1, \ldots, b_n\}$ as $\forall i, b_i = 2(a_i + 2n \max_k a_k)$. Thus, $b_i \geq \frac{\max_k b_k}{2}$, b_i is even. Moreover, if we let $M = \max_k b_k$ and $S = \frac{\sum_i b_i}{2}$, then $S \geq 100M$. We build the following instance of the (scaled) SSP problem (SSP$(b_1, \ldots, b_n)$): is there a partition of the $(20S + 17M) \times (20S + 17M)$ square into $14 + n + \sum_k b_k(M - b_k)$ squares of respective areas

$$\begin{aligned}
A_{13,11} &= (13S + 11M)^2 \quad (\times 1), \\
A_{7,6} &= (7S + 6M)^2 \quad (\times 3), \\
A_{4,3} &= (4S + 3M)^2 \quad (\times 2), \\
A_{3,3} &= (3S + 3M)^2 \quad (\times 2), \\
A_{3,2} &= (3S + 2M)^2 \quad (\times 2), \\
A_{2,2} &= (2S + 2M)^2 \quad (\times 4), \\
A_{bi} &= b_i^2 \quad (\forall i, 1 \leq i \leq n), \\
A_{1,1} &= 1 \quad (\times \sum_k b_k(M - b_k))?
\end{aligned}$$

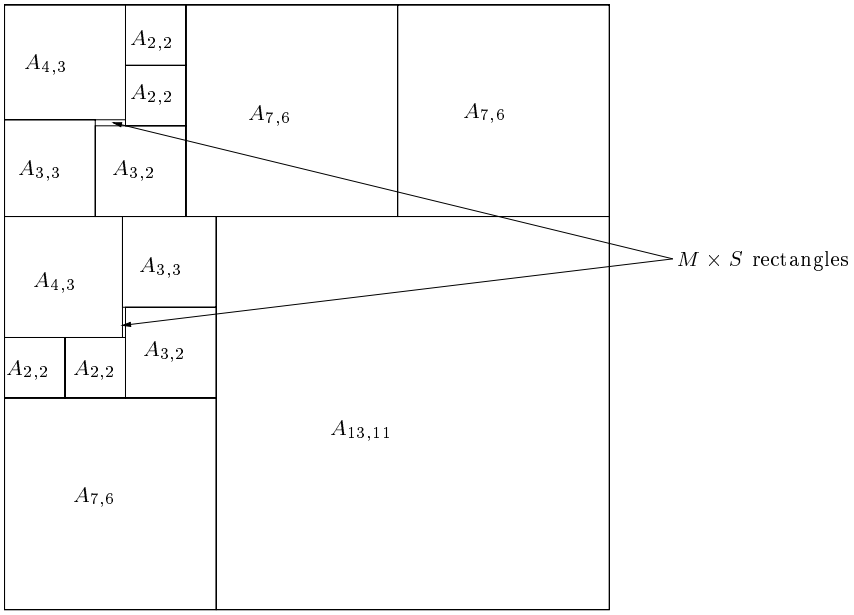


Figure 3: General position of the squares

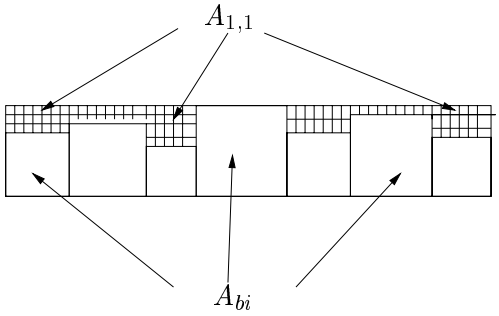


Figure 4: Zoom on the $M \times S$ rectangle areas

In what follows, we prove that such a partition is necessarily the one depicted in Figure 3, where the two small $M \times S$ rectangle areas are shown by arrows in Figure 3 and fully described in Figure 4. The intuitive idea of the proof is the following: the large squares are used to prevent the two small $M \times S$ rectangle areas to be neighbors. Hence these areas must be filled separately by the remaining squares, those of area b_i and those of area 1. This will be possible iff there the b_i's can be partitioned into two subsets of same cardinal and same sum. The number of squares of area 1 is computed so as to fill the holes and obtain a true tiling of the whole area.

Position of the Largest Four Squares The general position of the largest four squares is shown in Figure 5a. Obviously, if we can tile the remaining area with the remaining squares, this will also be the case for the configuration shown in Figure 5b. Therefore, from now on, we assume (without loss of generality) that the largest four rectangles are arranged as shown in Figure 5b.

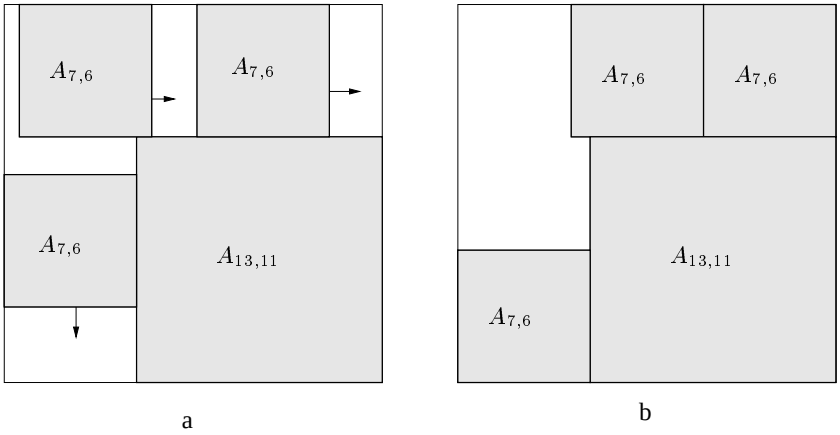


Figure 5: General position of the largest four squares

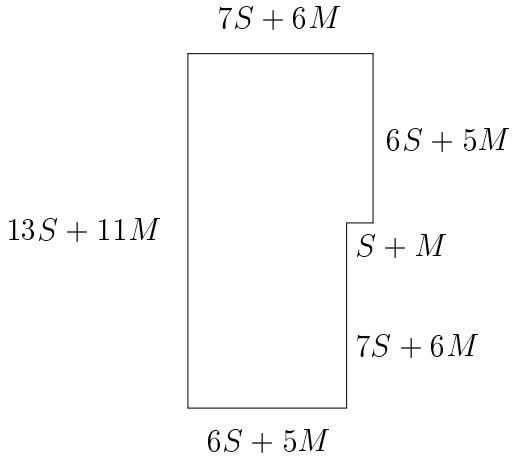


Figure 6: Remaining surface

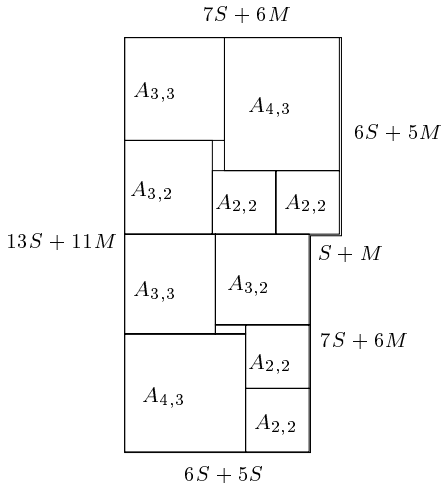


Figure 7: One possible tiling of the remaining surface.

Tiling the Remaining Surface Now we discuss the tiling of the remaining surface (the white area of Figure 5b). We give all dimensions in Figure 6). In the following figures, $A_{x,y}$ denotes a square of size $xS + yM$. Proceeding by an exhaustive case analysis, we prove that the only correct configuration is the one depicted in Figure 7 (other equivalent solutions are also possible).

Therefore, any tiling of the remaining surface (see Figure 6) is similar to the one depicted in Figure 7: after using all the large rectangles $A_{x,y}$, there remains two non-adjacent rectangle areas of area $M \times S$ to be tiled. Therefore, we can solve the SSP problem iff we can tile these two areas with the remaining squares, i.e. n squares of area $b_i^2, 1 \leq i \leq n$, and $\sum_k b_k(M - b_k)$ squares of area 1. Since $\min_k b_k \geq \frac{\max b_k}{2}$ and $\sum_k b_k = 2S$, one can easily check that both $M \times S$ rectangle areas have to be tiled as depicted in Figure 4. Therefore, our instance of the SSP problem has a solution iff there exists a partition of $\{b_1, \ldots, b_n\}$ into two subsets of same sum.

Final reduction To complete the reduction, we have to show that there exists a partition of $\{b_1, \ldots, b_n\}$ into two subsets of same sum iff the original instance of the 2-Partition-Equal problem has a solution.

First, suppose that the original instance of the 2-Partition-Equal problem has a solution, i.e. there exists a partition of $\{1, \ldots, n\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ satisfying

$$\sum_{k \in \mathcal{A}_1} a_k = \sum_{k \in \mathcal{A}_2} a_k \text{ and } \text{card}(\mathcal{A}_1) = \text{card}(\mathcal{A}_2).$$

Recall that $b_k = 2(a_k + 2n \text{ MAX})$, where $\text{MAX} = \max_k a_k$. Then,

$$\begin{aligned} \sum_{k \in \mathcal{A}_1} b_k &= \sum_{k \in \mathcal{A}_1} a_k + 2n \text{ MAX } \text{card}(\mathcal{A}_1) \\ &= \sum_{k \in \mathcal{A}_2} a_k + 2n \text{ MAX } \text{card}(\mathcal{A}_2) \\ &= \sum_{k \in \mathcal{A}_2} b_k \end{aligned}$$

Therefore, there exists a suitable partition of $\{b_1, \ldots, b_n\}$.

Conversely, suppose that there exists a partition of $\{1, \ldots, p\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that

$$\sum_{k \in \mathcal{A}_1} b_k = \sum_{k \in \mathcal{A}_2} b_k.$$

Thus,

$$\begin{aligned} \sum_{k \in \mathcal{A}_1} a_k &= \sum_{k \in \mathcal{A}_1} b_k - 2n \text{ MAX card}(\mathcal{A}_1) \\ \sum_{k \in \mathcal{A}_2} a_k &= \sum_{k \in \mathcal{A}_2} b_k - 2n \text{ MAX card}(\mathcal{A}_2) \\ \sum_{k \in \mathcal{A}_1} a_k - \sum_{k \in \mathcal{A}_2} a_k &= 2n \text{ MAX card}(\mathcal{A}_2 - \mathcal{A}_1) \end{aligned}$$

Moreover, since

$$\sum_{a_k \in \mathcal{A}_1} a_k - \sum_{a_k \in \mathcal{A}_2} a_k \leq n \text{ MAX},$$

we obtain

$$\text{card}(\mathcal{A}_1) = \text{card}(\mathcal{A}_2) \text{ and } \sum_{k \in \mathcal{A}_1} a_k = \sum_{k \in \mathcal{A}_2} a_k$$

Therefore, the original instance of the 2-Partition-Equal problem has a solution.

The last element of the proof is the conciseness of the transformation: we have to proof that our instance of the SSP problem has a size polynomial in the size of the original instance of the 2-Partition-Equal problem.

Lemma 3 *Define* $MAX = \max_k a_k$ *as above, and let* $c(a)$ *and* $c(b)$ *denote respectively the encoding of the data* a *and* b*. Then,*

$$Length(c(b)) = O(Length(c(a))^2).$$

This achieves the proof of the NP-completeness of SSP, and therefore of the NP-completeness of PERI-SUM. ■

3.3 PERI-MAX(s)

The decision problem associated to the optimization problem PERI-MAX is the following:

Definition 6 *PERI-MAX(s,K): Given* p *real positive numbers* $s_1, \ldots, s_p$ *s.t.* $\sum_{i=1}^{p} s_i = 1$ *and a positive real bound* K*, is there a partition of the unit square into* p *rectangles* R_i *of area* s_i *and of size* $h_i \times v_i$*, so that* $\max_{1 \leq i \leq p}(h_i + v_i) \leq K$*?*

Our second result states the intrinsic difficulty of the PERI-MAX optimization problem:

Theorem 2 *PERI-MAX(s,K) is NP-complete.*

Proof We give the main ideas of the proof, which uses the following reduction:

Lemma 4

$$\textit{2P-0-4} \leq_P \textit{MSP} \leq_P \textit{PERI-MAX},$$

where MSP and 2P-0-4 are defined as follows:

Definition 7 *2-Partition-0-4 (2P-0-4)*
Given a set of $p+2$ integers $\mathcal{A} = \{a_1, \ldots, a_p, a_{p+1} = 2, a_{p+2} = 2\}$ such that $(\forall i \leq p, \quad a_i > 4$ and $a_i = 0 \bmod 4)$, is there a partition of $\{1, \ldots, p+2\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that

$$\sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i \text{ or } \sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i + 4 \quad ?$$

Definition 8 *Max-Square-Partition (MSP)*
Given a set $\mathcal{A} = \{s_1, \ldots, s_p\}$ of p real positive numbers such that $\sum_{i=1}^{p} s_i = 1$ and $s_1 \geq s_2 \geq \ldots \geq s_p$, is there a partition of the unit square into p rectangles R_i of area s_i such that R_1 is a square and the half-perimeter of other rectangles is not larger than $2\sqrt{s_1}$?

Since 2P-0-4 is known to be NP-Complete (trivial reduction from 2-Partition [6]), Lemma 4 will complete the proof of Theorem 2.

3.3.1 Reduction: MSP $\leq_P$ PERI-MAX(s,K)

We start by proving the easy part of Lemma 4, i.e. MSP $\leq_P$ PERI-MAX(s,K). Let $\mathcal{A} = \{s_1, \ldots, s_p\}$ be a set of p real positive numbers s.t. $\sum_{i=1}^{p} s_i = 1$ and $s_1 \geq s_2 \geq \ldots \geq s_p$. Solving MSP is equivalent to solving PERI-MAX(s,K) with

$$K = 2\sqrt{s_1}$$

and therefore,

$$\text{MSP} \leq_P \text{PERI-MAX}.$$

3.3.2 Reduction: 2P-0-4 $\leq_P$ MSP

In this section, we consider an arbitrary instance of the 2-Partition-0-4 problem, i.e. a set $\mathcal{A} = \{a_1, \ldots, a_n, a_{n+1} = 2, a_{n+2} = 2\}$ such that $(\forall i \leq p, \quad a_i > 4$ and $a_i = 0 \bmod 4)$. We have to polynomially transform this instance into an instance of the MSP problem which has a solution iff the original instance of 2-Partition-0-4 has one solution. Let

$$S = \frac{\sum_{1 \leq i \leq n} a_i}{4}.$$

We build the following instance of the (scaled) MSP problem (MSP$(a_1, \ldots, a_n)$): is there a partition of the $(S+2) \times (S+2)$ square into $n+3$ rectangles $R_S, R_1, \ldots, R_{n+2}$ of respective areas

$$\begin{aligned} R_S: \quad A_S &= S^2, \\ R_i: \quad A_i &= a_i \qquad (\forall i, 1 \leq i \leq n), \\ R_{n+1}: \quad A_{n+1} &= 2, \\ R_{n+2}: \quad A_{n+2} &= 2, \end{aligned}$$

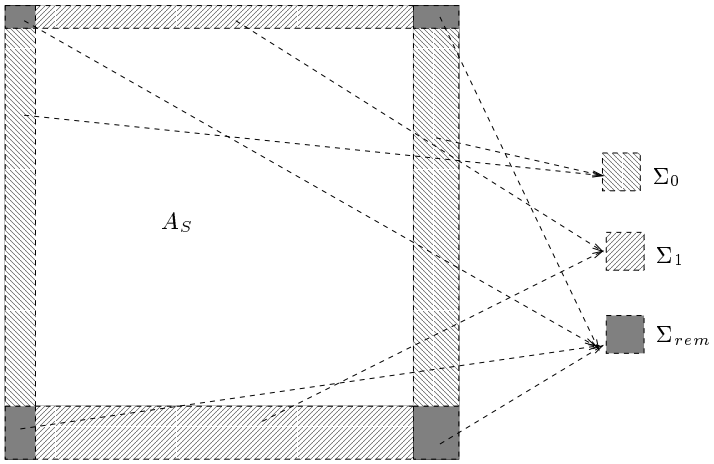


Figure 8: General position of the largest square.

where the rectangle R_S of area A_S is a square and the half-perimeter of other rectangles R_i is less than $2S$?

The general position of the largest square is depicted in Figure 8. We partition the set of the remaining rectangles into three disjoints sets:

- S_0: the rectangles whose intersection with Σ_0 has a non-zero area.
- S_1: the rectangles whose intersection with Σ_1 has a non-zero area.
- S_{rem}: the rest of the rectangles.

Since the area of Σ_{rem} is equal to 4, we can easily prove the following lemma:

Lemma 5 *$\forall i \leq n, R_i$ belongs either to S_0 or to S_1 ($a_i > 4$). Moreover, we have $|\sum_{R_i \in S_0} A_i - \sum_{R_i \in S_1} A_i| \leq 4$, i.e. $|\sum_{R_i \in S_0} A_i - \sum_{R_i \in S_1} A_i| \in \{0, 2, 4\}$.*

Without lost of generality, we suppose in what follows that $(\sum_{R_i \in S_0} A_i \geq \sum_{R_i \in S_1} A_i)$. Three different cases are to be considered according to the value of $(\sum_{R_i \in S_0} A_i - \sum_{R_i \in S_1} A_i)$:

- $(\sum_{R_i \in S_0} A_i - \sum_{R_i \in S_1} A_i) = 0$. In this case, either $\exists i$, $R_{n+1} \in S_i$ and $R_{n+2} \in S_{1-i}$, or both R_{n+1} and R_{n+2} belong to S_{rem}. In the first case, S_0 and S_1 represent a partition of $\mathcal{A} = \{a_1, \ldots, a_n, a_{n+1} = 2, a_{n+2} = 2\}$ into two subsets of same sum. In the second case $S_0 \bigcup R_{n+1}$ and $S_1 \bigcup R_{n+2}$ represent a partition of $\mathcal{A}$ into two subsets of same sum.
- $(\sum_{R_i \in S_0} A_i - \sum_{R_i \in S_1} A_i) = 2$. In this case, exactly one rectangle out of R_{n+1} and R_{n+2} belongs to S_0 or S_1, and the other one belongs to S_{rem}. Let us suppose, without loss of generality, that $R_{n+1} \in S_i$. Then S_0 and $S_1 \bigcup R_{n+2}$ represent a partition of $\mathcal{A}$ into two subsets of same sum.
- $(\sum_{R_i \in S_0} A_i - \sum_{R_i \in S_1} A_i) = 4$. Again, in this case, either $\exists i$, $R_{n+1} \in S_i$ and $R_{n+2} \in S_{1-i}$, or both R_{n+1} and R_{n+2} belong to S_{rem}. In the first case, S_0 and S_1 represent a partition of $\mathcal{A}$ into two subsets whose sums differ by 4. In the second case S_0 and $S_1 \bigcup R_{n+1} \bigcup R_{n+2}$ represent a partition of $\mathcal{A}$ into two subsets of same sum.

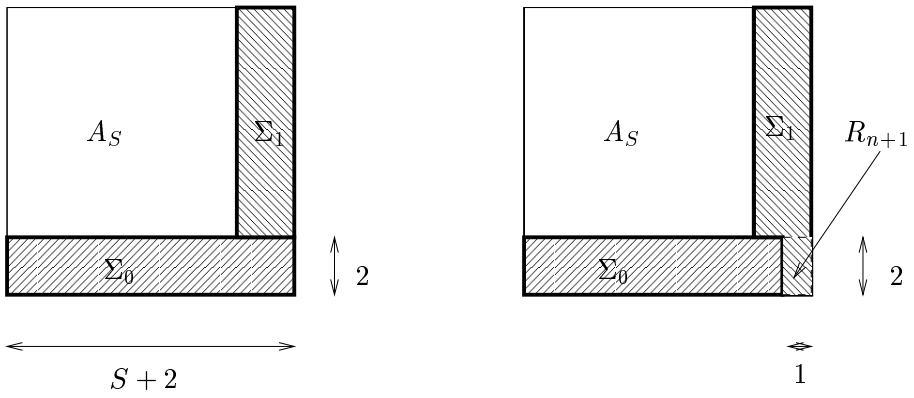


Figure 9: Tiling the square using the MSP instance.

Therefore, there exists a partition of $\mathcal{A} = \{a_1, \ldots, a_n, a_{n+1} = 2, a_{n+2} = 2\}$ into two subsets whose sums differ by 0 or 4 if our instance of the MSP problem has a solution. To complete the reduction, we have to show that our instance of the MSP problem has a solution if there exists a partition of $\mathcal{A} = \{a_1, \ldots, a_n, a_{n+1} = 2, a_{n+2} = 2\}$ into two subsets whose sums differ of 0 or 4.

- Suppose there is a partition of $\{1, \ldots, n+2\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that

$$\sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i + 4.$$

  Let us define $S_0 = \bigcup_{i \in \mathcal{A}_1} R_i$ where R_i denotes a $2 \times \frac{a_i}{2}$ rectangle, and $S_1 = \bigcup_{i \in \mathcal{A}_2} R_i$, where R_i denotes a $\frac{a_i}{2} \times 2$ rectangle. We tile the $(S+2) \times (S+2)$ area as indicated in Figure 9a. Since it is possible to tile both Σ_0 and Σ_1 with S_0 and S_1, it is possible to solve the MSP problem.

- Suppose there is a partition of $\{1, \ldots, n+2\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that

$$\sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i.$$

  Let us define $S_0 = \bigcup_{i \in \mathcal{A}_1} R_i$ where R_i denotes a $2 \times \frac{a_i}{2}$ rectangle, and $S_1 = \bigcup_{i \in \mathcal{A}_2} R_i$, where R_i denotes a $\frac{a_i}{2} \times 2$ rectangle. Suppose, without loss of generality that R_{n+1} belongs to S_1. Then, we tile the $(S+2) \times (S+2)$ area as indicated in Figure 9b. Hence, since it is possible to tile both Σ_0 and Σ_1 with S_0 and S_1 as depicted in Figure 9b, it is possible to solve the MSP problem.

Therefore, our instance of the MSP problem has a solution iff there exists a partition of $\mathcal{A} = \{a_1, \ldots, a_n, a_{n+1} = 2, a_{n+2} = 2\}$ into two subsets whose sums differ by 0 or 4. This achieves the proof of the NP-completeness of MSP, and therefore of the NP-completeness of PERI-MAX. ■

4 Approximation Algorithms for PERI-SUM

There are several "natural" heuristics to approximate PERI-SUM. However, proving approximation bounds turns out to be very technical. We start in Section 4.1 with a column-based heuristic, very simple to implement, and which appears very efficient through extensive experimental comparisons. However, we have not been able to give a tight approximation bound: the bound of Section 4.1.3 depends on the relative size of the rectangles to be used in the tiling. In Section 4.2 we move to a recursive heuristic, much more complicated to describe, but for which a nice approximation bound is provided.

4.1 Column-Based Heuristic

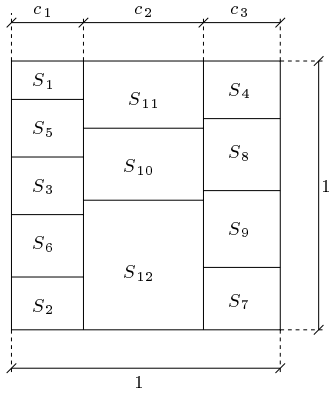


Figure 10: Column-based partitioning of the unit square: $\mathcal{C} = 3$, $k_1 = 5$, $k_2 = 3$ and $k_3 = 4$.

4.1.1 Description

Since PERI-SUM(s) is NP-complete, we consider the more constrained problem COL-PERI-SUM(s) where we impose that the tiling is made up with processor columns, as illustrated in Figure 10. In other words, COL-PERI-SUM(s) is the restriction of PERI-SUM(s) to column-based partitions. In this section, we give a polynomial solution to COL-PERI-SUM(s), which will be used as a heuristic for PERI-SUM(s).

Framework We describe the COL-PERI-SUM(s) problem more formally: we aim at tiling the unit square into $\mathcal{C}$ columns (where $\mathcal{C}$ is yet to be determined) of width $c_1, \ldots, c_{\mathcal{C}}$. Each column C_i is partitioned itself into k_i rows (to be determined too) of respective area $s_{\sigma(i,1)}, \ldots, s_{\sigma(i,k_i)}$. Of course, the final partitioning has $\sum_{i=1}^{\mathcal{C}} k_i = p$ rectangles, and all the areas $s_1, \ldots, s_p$ are represented once and only once. The goal is to build such a partitioning, subject to the minimization of the sum of the rectangle perimeters.

Algorithm The main points of the column-based tiling are the following:

1. Re-index the variables $s_1, \ldots, s_p$ such that $s_1 \leq s_2 \leq \ldots \leq s_p$.
2. Iteratively build the function $f_{\mathcal{C}}$, by incrementing the value of $\mathcal{C}$ from 1 to the desired value For $q \in \{1, \ldots, p\}$, $f_{\mathcal{C}}(q)$ represents the total perimeter of an optimal column-based partitioning

of a rectangle of height 1 and width $(\sum_{i=1}^{q} s_i) \times 1$ into q rectangles of respective area $s_1, \ldots, s_q$, using $\mathcal{C}$ columns.

To help understand the derivation, we apply the algorithm on the following toy example: we have $p = 10$ areas of values $(0.02, 0.04, 0.06, 0.08, 0.2, 0.2, 0.2, 0.2)$. The results of the algorithm are described in Table 1. Each column $\mathcal{C}_i$ contributes to the sum of the half perimeters as follows: 1 for the vertical line, and $k_i \times c_i$ for the k_i horizontal lines of length c_i.

	q=1	q=2	q=3	q=4	q=5	q=6	q=7	q=8
$\mathcal{C} = 1$	1.02 / 0	1.12 / 0	1.36 / 0	**1.8 / 0**	3 / 0	4.6 / 0	6.6 / 0	9 / 0
$\mathcal{C} = 2$		2.06 / 1	2.18 / 2	2.4 / 2	2.92 / 3	**3.6 / 4**	4.6 / 4	5.8 / 5
$\mathcal{C} = 3$			3.12 / 2	3.26 / 3	3.6 / 4	4.12 / 5	4.72 / 5	**5.4 / 6**
$\mathcal{C} = 4$				4.2 / 3	4.46 / 4	4.8 / 5	5.32 / 6	5.92 / 7
$\mathcal{C} = 5$					5.4 / 4	5.66 / 5	6 / 6	6.52 / 7
$\mathcal{C} = 6$						6.6 / 5	6.86 / 6	7.2 / 7
$\mathcal{C} = 7$							7.8 / 6	8.06 / 7
$\mathcal{C} = 8$								9 / 7

Table 1: Table containing the values of the couples $f_{\mathcal{C}}(q)/r$ where $f_{\mathcal{C}}(q) = min_{q'} \left(1 + \left(\sum_{q'<i\leq q} s_i\right) \times (q - q') + f_{\mathcal{C}-1}(q')\right) = 1 + \left(\sum_{r<i\leq q} s_i\right) \times (q - r) + f_{\mathcal{C}-1}(r)$. Bold entries correspond to the optimal solution.

In the example, the optimal partitioning is obtained for 3 columns ($f_3(8) = 5.4$). The last column of width $c_3 = s_7 + s_8 = 0.4$ is made of two elements. The second column of width $c_2 = s_5 + s_6 = 0.4$ is also made of two elements. Then the first column of width $c_1 = s_1 + s_2 + s_3 + s_4 = 0.2$ is made of the smallest 4 elements. Figure 11 represents this partitioning.

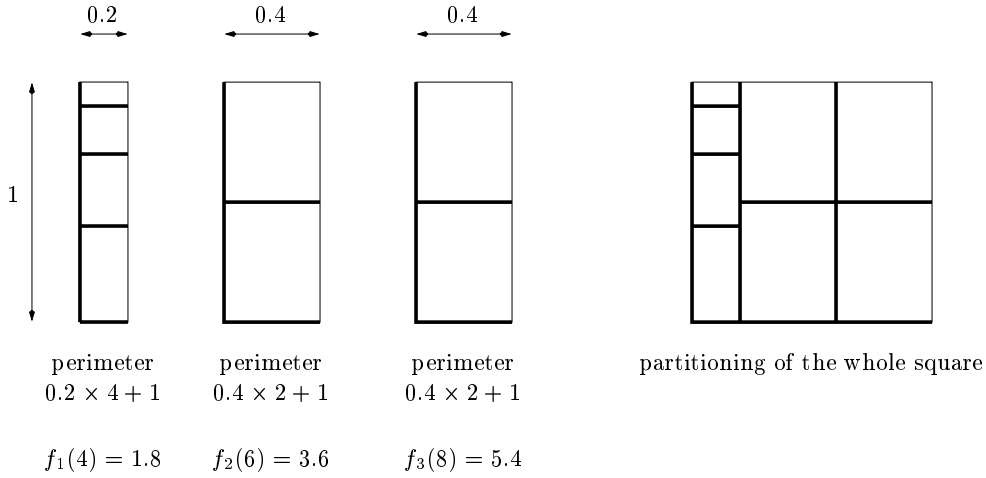


Figure 11: Optimal column-based partitioning for the example. Thicker lines correspond to the sum of the half perimeters.

The algorithm is outlined as follows:

$S = 0$
for q=1 to p
  $S = S + s_q$
  $f_1^{perimeter}(q) = 1 + S \times q$
  $f_1^{cut}(q) = 0$
endfor
for $\mathcal{C}$=2 to p
  for q=$\mathcal{C}$ to p
    $f_{\mathcal{C}}^{perimeter}(q) = \min_{1 \leq r \leq q-\mathcal{C}+1} \left(1 + S \times r + f_{\mathcal{C}-1}^{perimeter}(q-r)\right)$
    $f_{\mathcal{C}}^{cut}(q) = q - r_{min}$
  endfor
endfor

The worst-case complexity of the algorithm is $O(p^2 log(p))$: indeed, $f_{\mathcal{C}+1}(q)$ can be built from $f_{\mathcal{C}}$ in $0(\log p)$ steps, because the minimum in the algorithm can be searched by dichotomy: since for each $\mathcal{C}$, $f_{\mathcal{C}-1}$ is an non-decreasing function, $g_{\mathcal{C},q}(r) = 1+S\times r+f_{\mathcal{C}-1}^{perimeter}(q-r)$ is a convex function of r. Hence, the minimum $\min_{1 \leq r \leq q-\mathcal{C}+1}(g_{\mathcal{C},q}(r))$ can be found by dichotomy in $O(\log(q-\mathcal{C}+1)) = O(\log p)$ steps. Note that in practice the complexity will be lower than the worst-case analysis shows, because $f_{\mathcal{C}}(p)$ is a function that is first decreasing and then increasing as $\mathcal{C}$ varies. All the functions $f_{\mathcal{C}}$ will not be built, the expected cost will be $p\mathcal{C}_{min} \log(p) \approx p\sqrt{p} \log(p)$.

The final partitioning corresponding to the function $f_{\mathcal{C}_{min}}(p) = \min_{1 \leq \mathcal{C} \leq p} f_C(p)$ is found using the following algorithm:

$q = p$
for $\mathcal{C} = \mathcal{C}_{min}$ **downto** 2
  $k_{\mathcal{C}} = q - f_{\mathcal{C}}^{cut}(q)$
  $q = f_{\mathcal{C}}^{cut}(q)$
endfor
$k_1 = q$

which corresponds to tracking (backwards) the bold entries in Table 1. The unit square is partitioned into $\mathcal{C}_{min}$ columns. The i^{th} column contains the rectangles $s_d, s_{d+1}, \ldots, s_{d+k_i}$ with $d = k_1 + k_2 + \ldots + k_{i-1}$.

Optimality This algorithm provides the optimal column-based partitioning. The proof is detailed in [3]. The only difficulty is to prove that we can reduce the search to sorted sequences $s_1 \leq s_2 \ldots \leq s_p$.

4.1.2 Experimental Comparison with the Lower Bound

As shown in Section 3, a lower bound for the sum $\hat{C}$ of the half-perimeters is twice the sum of the square roots of the areas, i.e. $LB = 2\sum_{i=1}^{p} \sqrt{s_i}$. Of course this bound cannot always be met: consider an instance of PERI-SUM(s) with only two rectangles, $s_1 = 1 - \epsilon$ and $s_2 = \epsilon$, where $\epsilon > 0$ is an arbitrarily small number. Partitioning into two rectangles requires to draw a line of length 1, hence $\hat{C} = 3$. However, $LB = 2\sqrt{1-\epsilon} + \sqrt{\epsilon} > 2$ can be arbitrarily close to 2.

In this section, we experimentally compare, using a large number of random tests, the value $\hat{C}$ given by our partitioning against the absolute lower bound LB. Figure 12 represents two curves for a number of processors varying from 1 to 40. The first curve corresponds to the mean value of

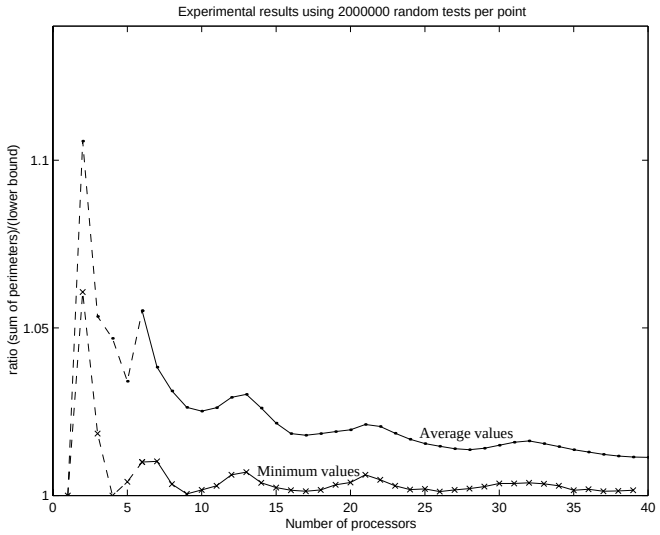


Figure 12: For each number of processors (varying from 1 to 40), 2,000,000 values for the s_i have been generated. For each case, we compute the ratio of the sum $\hat{C}$ of the half perimeters of our partitioning over the absolute lower bound LB. The worst case is a constant value equal to 1.5. The average and best cases are reported in the two curves.

the ratio $\frac{\hat{C}}{LB}$ while the second curve gives the minimum values of this ratio. We see that in average, the optimal column-based tiling given by our algorithm gives a solution that is "almost" optimal, so that we can be satisfied with the results for all practical purposes.

4.1.3 Theoretical Comparison with the Lower Bound

In this section, we prove that the column-based partitioning is a good approximation, especially when the ratio between $\max s_i$ and $\min s_i$ is small:

Proposition 1 *Let $r = \frac{\max s_i}{\min s_i}$, let $\hat{C}$ denote the sum of the half perimeters of the rectangles obtained with the optimal column-based partitioning, and let $LB = 2\sum_{i=1}^{p} \sqrt{s_i}$. Then,*

$$\frac{\hat{C}}{LB} \leq \sqrt{r}(1 + \frac{1}{\sqrt{p}})$$

The proof can be found in [3]. It is straightforward when evaluating the cost of a very simple partitioning (with about $\sqrt{p}$ columns and $\sqrt{p}$ elements per column). If $r = 1$, i.e. all the processors have the same speed, the column-based partitioning is asymptotically optimal. On the other hand, if r is large, i.e. if one rectangle is much larger than another, the bound is *very* pessimistic.

4.2 Recursive Heuristic

In this section we give a recursively defined heuristic, that will lead to a good approximation factor. We introduce this heuristic in two steps: first we define a column-based tiling which is different from that of the previous section; the idea here is to impose some ratio on the shapes of the rectangles. This column-based tiling is then used as a building block in the second step, where we derive the final tiling.

4.2.1 Column-Based Partitioning

We aim at tiling a rectangle R (not a square!) of size $h \times v$ such that $h \leq v \leq 4h$ into p rectangles of areas $s_1 \geq s_2 \geq \ldots \geq s_p$, where $\sum_{i=1}^{p} s_i = hv$. We further assume that $r = \frac{s_1}{s_p} \leq 2$. In this section, we show that there exists a column-based tiling of R into rectangles of area $s_i = h_i \times v_i$ such that

$$(CR): \ \frac{1}{4}v_i \leq h_i \leq 4v_i \qquad \forall i, 1 \leq i \leq p.$$

Note that this condition is equivalent to $\frac{\sqrt{s_i}}{2} \leq h_i, v_i \leq 2\sqrt{s_i}$. For convenience, we define:

$$\begin{cases} (CR)_h & \text{the condition} \quad h_i \geq \frac{1}{4}v_i \\ (CR)_v & \text{the condition} \quad h_i \leq 4v_i \end{cases}$$

The algorithm consists in two main phases:

First_phase
  $i = 1$
  $C_i = \{s_1\}$
  for j=2 **to** p
    if $(CR)_v$ is reached for all elements of C_i
      $i = i + 1$
      $C_i = s_j$
    endif
    else $C_i = C_i \cup s_j$
  endfor
  $\#columns = i$
endFirst_phase

Second_phase *Let* $s'_1 \geq s'_2 \ldots \geq s'_l$ *be the elements of the last column* C_i.
  if $(CR)_v$ is not reached by any element of C_i
    $\forall j \in \{1, \ldots, l\}$, $C_{j+i-l-1} = C_{j+i-l-1} \cup s'_j$
    $C_i = \emptyset$
    $\#columns = i - 1$
  endif
endSecond_phase

Figure 13 represents the partitioning of a 3×3.6 rectangle obtained with the following 7 rectangle areas: $(2, 2, 1.9, 1.5, 1.2, 1.2, 1)$. The proof of the correctness of the algorithm (that condition (CR) holds at the end) follows from the following three statements:

- [*Initial* $(CR)_h$]: consider any column (even the last one) after the first phase. Then, any element in this column fulfills condition $(CR)_h$.
- [*Final* $(CR)_h$]: the condition $(CR)_h$ still holds if we add one element to any column.
- [*Enough columns*]: assume that there are $c+1$ columns after the first phase. If the l elements in the last column do not fulfill condition $(CR)_v$, then $l \leq c$.

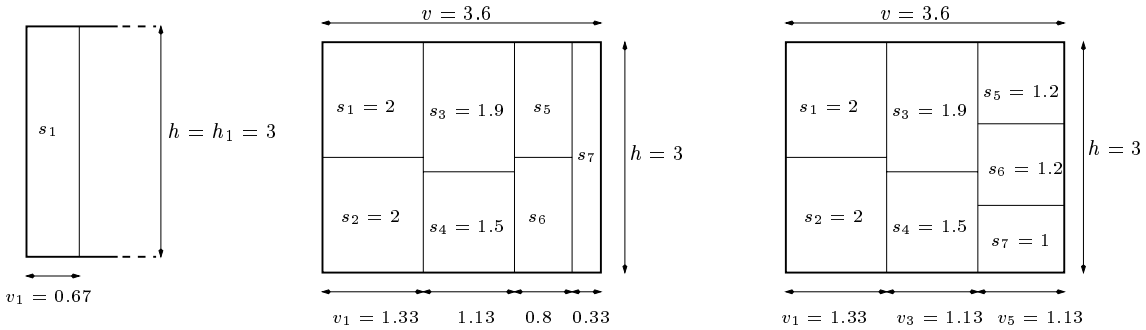


Figure 13: The first two figures correspond to the first phase of the algorithm: columns are filled using $(CR)_v$ as a stopping criterion: for the first column, leaving s_1 alone would not fulfill the condition $(CR)_v$. Indeed, $\frac{h_1}{4} = \frac{3}{4} = 0.75 > 0.67 = v_1$. Another element must be added to the column. Thus, we get $h_1 = h_2 = 1.5$ whereas $v_1 = 1.33$. The condition is then fulfilled and the algorithm goes through the next column. The last figure corresponds to the second phase of the algorithm: since the elements of the last column do not fulfill condition $(CR)_v$, they are distributed over other columns (in this case, only one element ($s_7 = 1$) has to be distributed).

Indeed, consider a column composed of k elements $s'_1 \geq \ldots \geq s'_k$. Then for each element $s'_i = h'_i \times v'_i$ of this column, we have $h'_i = \frac{s'_i}{\sum_{j=1}^{k} s'_j} \times h$ and $v'_i = \frac{\sum_{j=1}^{k} s'_j}{h}$. Consequently, we have $(CR)_h$: $\sqrt{s'_k} \geq \frac{1}{2}\frac{\sum_{j=1}^{k} s'_j}{h}$ and $(CR)_v$: $\sqrt{s'_1} \leq 2\frac{\sum_{j=1}^{k} s'_j}{h}$. In particular, we see that as soon as $(CR)_v$ holds true for the elements of a column, then it will still hold true if we add a new element to this column.

Initial $(CR)_h$ Let us denote by $s'_1 \geq s'_2 \geq \ldots \geq s'_k$ the elements of the column. Let v'_1 denote its width and $\forall 1 \leq i \leq k$, h'_i the height of the rectangle of area s'_i. Two situations may occur:

1. $[k = 1]$: we have $v'_1 \leq v \leq 4h = 4h'_1$
2. $[k > 1]$: because $(CR)_h$ does not hold if we remove the k^{th} element, we have $\forall i \in \{1, \ldots, k\}$, $\frac{\sum_{j=1}^{k-1} s'_j}{h} < \frac{1}{2}\sqrt{s'_1} < \frac{1}{\sqrt{2}}\sqrt{s'_i}$. Consequently, $v'_1 = \frac{\sum_{j=1}^{k} s'_j}{h} < \frac{k}{k-1} \times \frac{1}{\sqrt{2}}\sqrt{s'_i} < 2\sqrt{s'_i}$.

Final $(CR)_h$ Let $s'_1 \geq s'_2 \geq \ldots \geq s'_k$ be the elements of a column, and let $s = s'_{k+1}$ the element to be added to this column. Taking into account the new element, we define the values v'_1 and h'_i as previously. Three different cases may occur:

1. $[k = 1]$: we show that in the worst case, there is only one element in the last column. Indeed, consider a column $s''_1, \ldots, s''_l$ of $l > 1$ elements. Then $v''_1 = \frac{\sum_{i=1}^{l} s''_i}{h} \geq \frac{l}{2}\frac{s'_1}{h} \geq \frac{1}{4}h'_1 = \frac{1}{4}h > \frac{1}{4}h''_i$. Consequently, $\frac{s}{h} < \frac{h}{4}$. But $s'_1 \leq 2s$. Hence, $v'_1 = \frac{s'_1+s}{h} < \frac{3}{4}h$. With two elements in a column, $h'_i \geq \frac{h}{3}$, so finally $v'_1 < \frac{9}{4}h'_i$.
2. $[k = 2]$: in this case, $\frac{s'_1}{h} < \frac{h}{4}$ and $s'_1 \geq s'_2 \geq s$. Thus, $v'_1 = \frac{s'_1+s'_2+s}{h} < \frac{3}{4}h$. If there are three elements in the column, then $h'_i \geq \frac{h}{5}$. Consequently, $v'_1 < \frac{15}{4}h'_i$.
3. $[k \geq 3]$: in this case $v'_1 = \frac{\sum_{i=1}^{k+1} s'_i}{h} < \frac{k+1}{k-1} \times \frac{\sum_{i=1}^{k-1} s'_i}{h} < \frac{k+1}{k-1} \times \frac{\sqrt{s'_1}}{2} < \sqrt{2}\sqrt{s'_i}$.

Enough columns Let v_i (for $1 \leq i \leq c+1$) be the width of the i-th column after the first phase. Let h'_i (for $1 \leq i \leq l$) denote the heights of the elements of the last column. We have shown above (see paragraph *initial* $(CR)_v$) that $\forall 1 \leq i \leq c,\ 1 \leq j \leq p,\ v_i < \sqrt{2}\sqrt{s_j}$. Since $(CR)_v$ is not fulfilled by the elements of the $(c+1)^{th}$ column we know that $v_{c+1} < \frac{1}{\sqrt{2}}\sqrt{s_j} < \sqrt{2}\sqrt{s_j}$ and $\forall 1 \leq i \leq l,\ 1 \leq j \leq p,\ h'_i > \sqrt{2}\sqrt{s_j}$. Consequently, $\forall 1 \leq i \leq c+1,\ 1 \leq j \leq l,\ v_i < h'_j$. Moreover, since $v \geq h$, $\sum_{i=1}^{c+1} v_i = v$ and $\sum_{j=1}^{l} h'_j = h$, it is clear that $l \leq c$. ■

Remark A consequence of this result is that we can improve the bound of Proposition 1 when $r \leq 2$: we get $\frac{\hat{C}}{LB} \leq \frac{5}{4}$.

4.2.2 Recursively defined partitioning

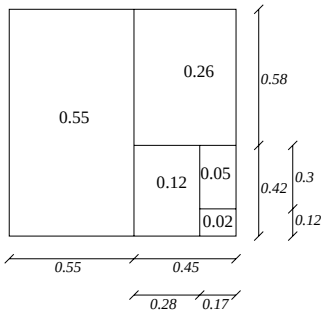
The main idea here is to split the list $s_1 \geq s_2 \geq \ldots \geq s_p$ of the rectangle areas into sub-lists so that the previous condition $r \leq 2$ holds within each sub-list. For instance, if $S = (0.49, 0.2, 0.2, 0.1, 0.01)$, then we get three sub-lists (0.49), $(0.2, 0.2, 0.1)$ and (0.01). Then we compute the sum of the elements within each sub-list. Considering those results as new input values, we get a smaller problem. In the example, we get $S = (0.5, 0.49, 0.01))$. Then, we restart the process recursively until no more merging is possible. In the example, we convergence after the third step, with $S = (0.99, 0.01))$. At the end of the process, since $\forall i,\ \ s_i > 2s_{i+1}$, the following inequality holds true:

$$\sum_{j>i} s_j < \frac{s_i}{2} + \frac{s_i}{4} + \ldots < s_i.$$

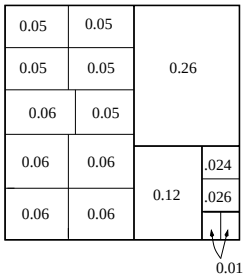
In what follows, we will denote by S_i the sub-lists obtained after convergence and by k_i the cardinal of S_i. The partitioning algorithm is recursively defined with two main functions, as outlined below:

Initial_square $(h, v, S = (s_1, \ldots, s_k))$
here necessary, the s_i should fulfill the condition $s_i > \sum_{j>i} s_j$
if $k > 1$
if $v \geq h$
partition $h \times v$ into $h \times v_1 = s_1$ and $h \times v_2 = \sum_{j>i} s_j$
compute **Column_based**(h, v_1, S_1) and **Initial_square**$(h, v_2, (s_2, \ldots, s_k))$
else
partition $h \times v$ into $h_1 \times v$ and $h_2 \times v$ and compute similarly
endif
endif
endInitial_square

Column_based $(v, h, S = (s_1, \ldots, s_k))$
here necessary, the s_i should fulfill the condition $\frac{1}{2} \leq \frac{s_i}{s_j} \leq 2$
if $\frac{1}{4} \leq \frac{v}{h} \leq 4$
apply step 1 and 2 of the algorithm of section 4.2.1
else
apply step 1 only of the algorithm of section 4.2.1
endif
$\forall i$ such that $k_i > 1$, compute **Column_based**(v_i, h_i, S_i)
endColumn_based



(a) Initial partitioning



(b) Final partitioning

Figure 14: Partitioning the square with the following initial list of rectangles: $S = (0.26, 0.12, 0.06, 0.06, 0.06, 0.06, 0.06, 0.05, 0.05, 0.05, 0.05, 0.05, 0.026, 0.024, 0.01, 0.01)$. In this case, convergence is obtained after one step of the algorithm, and then $S = (0.55, 0.26, 0.12, 0.05, 0.02)$. The initial partitioning (a) is described in the left part of the figure, while the final partitioning (b) is shown on the right. Note that the rectangles of the initial partitioning have been recursively partitioned into smaller rectangles with the column-based algorithm.

Proposition 2 *Let $\hat{C}$ denote the sum of the half perimeters of the rectangles obtained with the recursively defined partitioning, and let $LB = 2\sum_{i=1}^{p}\sqrt{s_i}$. Then,*

$$\hat{C} \leq 1 + \frac{5}{4}LB$$

Proof We have to show that the additional cost to pay for rectangles that do not fulfill condition (CR) is less than 1. For the partitioning of a given rectangle, let us call this quantity *the extra-cost*. Depending on the depth of the recursion, two kinds of partitioning may arise:

- If *all* the elements differ by a ratio of *less* than 2, then a column based partitioning is used.
- If *all* the elements differ by a ratio of *more* than 2, then the rectangle is partitioned into two parts and the smallest part is itself partitioned into two parts recursively.

Consequently, the proof is made of two parts:

1. [*Extra-cost for the column based partitioning*] In that case, we show that the extra-cost for the partitioning of a rectangle of size $h \times v$ $(v \geq h)$ is less than $v - h$.
2. [*Extra-cost for the initial partitioning of the square*] In that case, we show that the extra-cost for the partitioning of a rectangle of size $h \times v$ $(v \geq h)$ is less than h.

Extra-cost for the column based partitioning

1. If there is only one rectangle, then the extra-cost is $v + h - 2\sqrt{vh} \leq v - h$.
2. Suppose that there are more than one rectangle and that the initial rectangle is so thin that its partitioning is made of one element only per column. In that case, for each rectangle, the extra-cost is less than $v_i - h$, so that the overall extra-cost is less than $\sum_{i=1}^{c} v_i - h = v - ch < v - h$.

3. Suppose that there are more than one rectangle per column so that the last column only is unbalanced (its elements do not fulfill condition (CR)). For each elements of the last column, the extra-cost is less than $h_i - v_c$. Hence, the overall extra-cost is less than $\sum_{i=1}^{l} h_i - v_c = h - lv_c < h$ and, since $4h < v$, less than $v - h$.

Extra-cost for the initial partitioning of the square Suppose without loss of generality that the $h \times v$ (where $v \geq h$) rectangle is partitioned into two rectangles $h \times v_1$ and $h \times v_2$ where $v_1 > v_2$. Thus, $h \times v_1$ is tiled with the column-based algorithm, so that the extra-cost for this rectangle is less than $v_1 - h$ if $v_1 > h$ and 0 otherwise, since $h \leq v < 2v_1$.
Two situation may occur for the remaining rectangle:

- If $v_2 \geq h$, its extra-cost is less than v_2, so $v_1 > v_2 \geq h$. Hence, the overall extra-cost is less than $v_1 - h + v_2 < v$.
- If $v_2 \leq h$, its extra-cost is less than h. So, either $v_1 \geq h$, then the overall extra-cost is less than $v_1 - h + h < v$; or $v_1 < h$, and then the overall extra-cost is less than $h \leq v$.

As a consequence, the extra-cost for the partitioning of the initial square is less than 1. ■

5 Approximation algorithms for PERI-MAX

In this section, we introduce a polynomial heuristic to solve the PERI-MAX problem. Again, we consider a column based partitioning of the unit square. We consider two different heuristics, according to the area of the largest rectangle. Let $s_1 \geq s_2 \ldots \geq s_p$ denote the given areas of the rectangles.

If s_1 is greater than $\frac{1}{3}$, we use a first heuristic. In this case, one column is created for each element. Therefore, the half-perimeter of one rectangle of area s_i is $1 + s_i$. In this case,

$$\forall 1 \leq i \leq p, r_i = \frac{1 + s_i}{2\sqrt{s_1}} \leq r_1 \leq \frac{2}{\sqrt{3}}.$$

In the case where s_1 is less than $\frac{1}{3}$, we use a second heuristic, which ensures that

$$\forall 1 \leq i \leq p, r_i = \frac{h_i + v_i}{2\sqrt{s_1}} \leq \frac{2}{\sqrt{3}}.$$

The algorithm can be stated as follows:

Peri-max_column-based $(p, S = (s_1, \ldots, s_p))$
$c = 1$
for $i = 1$ **to** p
Scol(c)=Scol(c) $\bigcup\{i\}$
if $\sum_{i \in Scol(c)} s_i \geq 2\sqrt{\frac{s_1}{3}} - \sqrt{\frac{4s_1}{3} - \max_{i \in Scol(c)} s_i}$.
$c = c + 1$
endif
endfor
$c_{max} = c$
if $\sum_{i \in Scol(c_{\max})} s_i \leq 2\sqrt{\frac{s_1}{3}} - \sqrt{\frac{4s_1}{3} - \max_{i \in Scol(c_{\max})} s_i}$
$Scol(1) = Scol(1) \bigcup Scol(c_{max})$
$c_{max} = c_{max} - 1$
endif
endPeri-max_column-based

The configuration of one of the columns Scol is depicted in Figure 15. The largest perimeter of the rectangles in the column Scol(c) is

$$\sum_{i \in Scol(c)} s_i + \frac{\max_{i \in Scol(c)} s_i}{\sum_{i \in Scol(c)} s_i}.$$

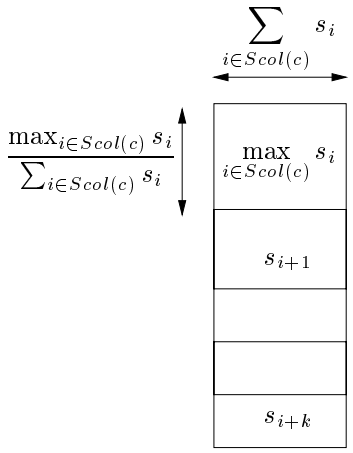


Figure 15: Scol(c).

As shown in Figure 16, the condition

$$\forall i \in Scol(c), \quad \max(h_i + v_i) \leq \frac{4\sqrt{s_1}}{\sqrt{3}}$$

holds true iff

$$2\sqrt{\frac{s_1}{3}} - \sqrt{\frac{4s_1}{3} - \max_{i \in Scol(c)} s_i} \leq \sum_{i \in Scol(c)} s_i \leq 2\sqrt{\frac{s_1}{3}} + \sqrt{\frac{4s_1}{3} - \max_{i \in Scol(c)} s_i}.$$

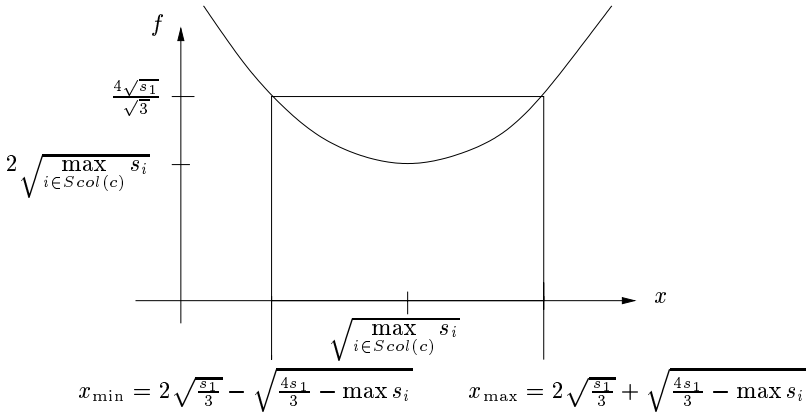


Figure 16: Plot of the function $f(x) = x + \frac{\max_{i\in Scol(c)} s_i}{x}$.

Therefore, the condition

$$\forall i \in Scol(c), \quad \max(h_i + v_i) \leq \frac{4\sqrt{s_1}}{\sqrt{3}}$$

holds true, except perhaps for the first column (since $Scol(c_{\max})$ have possibly been added). Indeed, since

$$x_{\max}(Scol(c)) - x_{\min}(Scol(c)) = 2\sqrt{\frac{4s_1}{3} - \max_{i\in Scol(c)} s_i} \geq \frac{2\sqrt{s_1}}{\sqrt{3}} \geq s_1,$$

we cannot jump from $x_{\min}(Scol(c))$ to $x_{\max}(Scol(c))$ by adding just one rectangle to $\mathrm{Scol}(c)$.

Hence, in order to prove the correctness of our algorithm, we need to prove the following two points:

- There are at least two columns at the end of the first step of the algorithm. Indeed,

$$\sum_{i\in\{1..n\}} s_i = 1 > \sqrt{s_1} \geq x_{\min}(Scol(1)).$$

- Suppose that the last column $\mathrm{Scol}(c_{\max})$ does not fulfill the condition

$$\forall i \in Scol(c_{\max}), \quad \max(h_i + v_i) \leq \frac{4\sqrt{s_1}}{\sqrt{3}}.$$

  Then, the condition

$$\forall i \in Scol(1), \quad \max(h_i + v_i) \leq \frac{4\sqrt{s_1}}{\sqrt{3}}$$

  still holds true if

$$Scol(1) = Scol(1) \bigcup Scol(c_{\max}).$$

  Indeed, in this case, we know that

$$\sum_{i\in Scol(c_{\max})} s_i \leq x_{\min}(Scol(c_{\max})) = 2\sqrt{\frac{s_1}{3}} - \sqrt{\frac{4s_1}{3} - \max_{i\in Scol(c_{\max})} s_i} \leq \sqrt{\frac{s_1}{3}}$$

and

$$\sum_{i \in Scol(1)} s_i \leq x_{\min}(Scol(1)) + s_1 \leq \sqrt{\frac{s_1}{3}} + s_1.$$

Therefore, since $s_1 \leq \frac{1}{3}$,

$$\sum_{i \in Scol(c_{\max}) \bigcup Scol(1)} s_i \leq 2\sqrt{\frac{s_1}{3}} + s_1 \leq \sqrt{3s_1} = x_{\max}(Scol(1)).$$

In summary, by using one of the two heuristics according to the value of s_1, we have proven the following proposition:

Proposition 3 *Let $\hat{M}$ denote the maximum of the half perimeters of the rectangles obtained with the above heuristic, and let $LB = 2\sqrt{s_1}$. Then,*

$$\frac{\hat{M}}{LB} \leq \frac{2}{\sqrt{3}}$$

Note that it is impossible to obtain a better guarantee (without taking into account the actual values of the s_i's). Indeed, if we consider the following situation with $s_1 = s_2 = s_3 = \frac{1}{3}$, then the optimal solution satisfies to

$$\hat{M} = \max_i (h_i + v_i) = \frac{2}{\sqrt{3}} 2\sqrt{s_1} = \frac{2}{\sqrt{3}} LB.$$

6 Related Results

In this section, we survey results on geometric optimization problems similar to PERI-SUM or PERI-MAX:

Covering a square by small perimeter rectangles Alon and Kleitman [1] consider the tiling of the unit square into n rectangles. There is no constraint on the area of the rectangles. They show that one of the rectangle must have perimeter at least $4(2m+1)/(n+m(m+1))$, where m is the largest integer whose square is at most n. This result is exact for $n = m(m+1)$ or $n = m^2$.

Decomposition of a square into rectangles of minimal perimeter Kong et al. [12] determine how to tile the unit square into p rectangles of same area so as to minimize the maximum perimeter of these rectangles. This is exactly our PERI-MAX problem constrained to same-area rectangles ($s_i = 1/p$ for $1 \leq i \leq p$). This problem is shown to be polynomial in [12]. The optimal solution is one of the following two arrangements: let either $m = \lfloor \sqrt{p} \rfloor$ or $m = \lceil \sqrt{p} \rceil$, and use m columns composed of $\lfloor \frac{n}{m} \rfloor$ or $\lceil \frac{n}{m} \rceil$ rectangles. This solution is extended to deal with the decomposition of a rectangle (instead of a square) onto same-area rectangles in [11].

Partitioning a rectangle with interior points Another related problem is to find the minimum partition of a rectangle with interior points: given a rectangle R and a finite set P of points located inside R, find a set of line segments that partition R into rectangles such that every point in P is on the boundary of some rectangle. The goal is to minimize the

total length of the introduced line segments. This problem is shown NP-complete in [13] and approximation algorithms are given in [7, 8]. The link with our PERI-MAX problem is that the objective function is the same, but the original motivation in [7, 8] was a VLSI routing problem (and the constraints are quite different).

Array partitioning The minimum rectangle tiling problem [9] is partially related to our optimization problems PERI-MAX: given an $n \times n$ array A of non-negative numbers, and a positive integer p, find a partition of A into p non-overlapping rectangular subarrays, such that the maximum weight of any rectangle in the partition is minimized (the weight of a rectangle is the sum of its elements). This problem is NP-complete, and approximation algorithms are given in [10, 9]

Finally, note that there are several NP-complete geometric optimization problems that are listed in the NP Compendium [5]. See also the survey book [2].

7 Conclusion

In this paper, we have dealt with two geometric problems arising from heterogeneous parallel computing. Because both problems have been shown NP-complete, we have introduced approximation algorithms.

The original motivation for this work is very important: the MMM algorithm is the prototype of tightly-coupled kernels that need to be implemented efficiently on distributed and heterogeneous platforms: we view it as a perfect testbed before experimenting more challenging computational problems on the grid.

References

[1] N. Alon and D.J. Kleitman. Covering a square by small perimeter rectangles. *Discrete Computational Geometry*, 1:1–7, 1986.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer, Berlin, Germany, 1999.

[3] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Matrix-matrix multiplication on heterogeneous platforms. Technical Report RR-2000-02, LIP, ENS Lyon, January 2000.

[4] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, 1997.

[5] P. Crescenzi and V. Kann. A compendium of NP optimization problems. World Wide Web document, URL: `http://www.nada.kth.se/~viggo/wwwcompendium/wwwcompendium.html`.

[6] Michael R. Garey and Davis S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1991.

[7] T.F. Gonzalez and S. Zheng. Improved bounds for rectangular and guilhotine partitions. *J. Symbolic Computation*, 7:591–610, 1989.

[8] T.F. Gonzalez and S. Zheng. Approximation algorithm for partitioning a rectangle with interior points. *Algorithmica*, 5:11–42, 1990.

[9] S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *Proc. 9th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 384–393. ACM Press, 1998.

[10] S. Khanna, S. Muthukrishnan, and S. Skiena. Efficient array partitioning. In *Proc. 24th Int. Colloquium on Automata, Languages and Programming*, LNCS 1256, pages 616–626. Springer-Verlag, 1997.

[11] T.Y. Kong, D.M. Mount, and W. Roscoe. The decomposition of a rectangle into rectangles of minimal perimeter. *SIAM J. Computing*, 17(6):1215–1231, 1988.

[12] T.Y. Kong, D.M. Mount, and M. Wermann. The decomposition of a square into rectangles of minimal perimeter. *Discrete Applied Mathematics*, 16:239–243, 1987.

[13] A. Lingas, R.Y. Pinter, R.L. Rivest, and A. Shamir. Minimum edge length partitioning of rectilinear polygons. In *Proc. 20th Ann. Allerton Conference on Communication, Control and Computing*, 1982.