Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 149 (2003) 533–539
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

Discrete Optimization

Complexity and approximability results for slicing floorplan designs

Vladimir G. Deĭneko [a], Gerhard J. Woeginger [b,c,*]

[a] Warwick Business School, The University of Warwick, Coventry CV4 7AL, UK
[b] Department of Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
[c] Institut für Mathematik, Technische Universität Graz, Steyrergasse 30, A-8010 Graz, Austria

Received 16 January 2002; accepted 7 June 2002

Abstract

The first stage in hierarchical approaches to floorplan design determines certain topological relations between the positions of indivisible cells on a VLSI chip. Various optimizations are then performed on this initial layout to minimize certain cost measures such as the chip area. We consider optimization problems in fixing the orientations of the cells and simultaneously fixing the directions of the cuts that are specified by a given slicing tree; the goal is to minimize the area of the chip.

We prove that these problems are NP-hard in the ordinary sense, and we describe a pseudo-polynomial time algorithm for them. We also present fully polynomial time approximation schemes for these problems.

Keywords: Packing; Cutting; Floorplan design; VLSI design; Compaction; Computational complexity; Approximation; Combinatorial optimization

1. Introduction

In hierarchical approaches to VLSI floorplan design (see for instance [4–6,8]), the first stage of the approach provides information on certain topological relations between the positions of cells and groups of cells on a chip. Here a *cell* is an indivisible rectangular entity on the chip that eventually will accomodate a VLSI component. Since this VLSI component possibly will be rotated, the exact dimensions of the cell are not fixed a priori. Instead, two positive integers a and b are associated with the cell that state the vertical and horizontal dimensions of the VLSI component that must fit into the cell. Then the final shape of that cell must either be a rectangle with integral width at least a and integral height at least b (in case the component is rotated) or a rectangle with integral width at least b and integral height at least a (in case the component is not rotated). We denote this set of compatible rectangular shapes for the cell by $\mathscr{R}(a,b)$. A *layout* of the chip is an enclosing rectangle that is subdivided by horizontal and vertical line segments into non-overlapping cells.

* Corresponding author. Tel.: +31-53-489-3462; fax: +31-53-489-4858.
E-mail addresses: orsvd@wbs.warwick.ac.uk (V.G. Deĭneko), g.j.woeginger@math.utwente.nl (G.J. Woeginger).

0377-2217/03/$ - see front matter
doi:10.1016/S0377-2217(02)00527-1

A *Guillotine cut* in a layout is a straight line that starts at one side of the rectangle, and then goes all the way through the rectangle to its opposite side. Throughout the paper, we will only consider *non-trivial* Guillotine cuts that split the rectangle into two non-empty pieces. In a general layout as defined above, the horizontal and vertical subdividing lines that generate the cells may be completely arbitrary. For instance, they may form a layout as depicted in Fig. 1(a) in which the reader will not find a single (non-trivial) Guillotine cut. A *floorplan* layout is a special type of layout that results from a sequence of Guillotine cuts. The first Guillotine cut splits the enclosing rectangle into two pieces. The following Guillotine cuts then split these pieces and their subpieces, and their sub-subpieces, and so on. See Fig. 1(b) for an illustration. Clearly, a floorplan is a particularly simple and attractive type of layout [6].

One way of representing floorplan layouts is via so-called *slicing trees*. A slicing tree (see Fig. 2 for an illustration) is a rooted binary tree where every interior vertex has exactly two children. Every leaf of a slicing tree corresponds to a rectangular cell for some VLSI component. Every interior vertex of a slicing tree corresponds to a (horizontal or vertical) Guillotine cut. The *depth* of a slicing tree T is the length of the longest path from the root of T to a leaf. We denote the number of leaves in T by $n = n(T)$, and the depth of T by $d = d(T)$. With every slicing tree T, the following family $\mathscr{F}(T)$ of floorplan layouts is associated.

- If the tree T consists of a single leaf that corresponds to a cell with width and height parameters a and b, then $\mathscr{F}(T) = \mathscr{R}(a, b)$.
- If the root of T has two children, then denote by T_ℓ and T_r the subtrees rooted at the left and the right child of the root, respectively. If the root is labeled by an 'h' (horizontal), then $\mathscr{F}(T)$ consists of all layouts that can be constructed by putting a layout from family $\mathscr{F}(T_\ell)$ below a layout from family $\mathscr{F}(T_\mathrm{r})$ and by separating them via a horizontal Guillotine cut. If the root is labeled by a 'v' (vertical), then $\mathscr{F}(T)$ consists of all layouts that can be constructed by putting a layout from family $\mathscr{F}(T_\ell)$ to the left of family $\mathscr{F}(T_\mathrm{r})$ and by separating them via a vertical Guillotine cut.

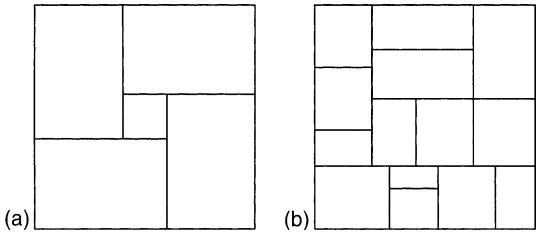

Fig. 1. (a) Not a floorplan layout. (b) A floorplan layout.

The usual goal in chip design is to minimize the area of the chip layout. Hence, the following basic optimization problem arises: given a slicing tree T, find a layout of smallest possible area in $\mathscr{F}(T)$. Since the orientations of the VLSI components in the leaves and the orientations of the vertical/horizontal Guillotine cuts in the interior vertices both may either be a priori fixed or left unspecified, we arrive at the following four basic variants of this optimization problem.

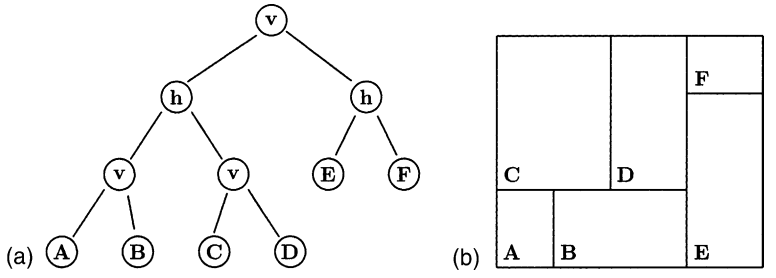

Fig. 2. (a) A slicing tree. (b) A corresponding layout.

(V1) The orientations of the cells are fixed, and the orientations of the Guillotine cuts are fixed. This problem is trivial, since there is nothing left to optimize, and there is only one candidate layout in $\mathscr{F}(T)$.

(V2) The orientations of the cells are unspecified, and the orientations of the Guillotine cuts are fixed. Stockmeyer [7] presented a polynomial time algorithm that finds the optimal layout for the slicing tree T within $O(nd)$ steps of computation.

(V3) The orientations of the cells are fixed, and the orientations of the Guillotine cuts are unspecified. To the best of our knowledge, this problem has not been investigated before.

(V4) The orientations of the cells are unspecified, and the orientations of the Guillotine cuts are unspecified. Almeida et al. [1] present an exact algorithm for this problem that finds the optimal layout for the slicing tree T with $O(2^n)$ steps of computation in the worst case. Computational experiments indicate that the algorithm in [1] performs very well in practice.

Results of this paper. In this paper, we will completely settle the computational complexity and the approximability of the floorplan layout problem variants (V3) and (V4). As a first (negative) result, we will show that both variants are NP-complete. On the one hand, this result explains why Almeida et al. [1] were not able to find a polynomial time algorithm for the problem variant (V4). On the other hand, this result draws a clear separating line between the hard variants (V3) and (V4) with unspecified orientations of the Guillotine cuts, and the easy variants (V1) and (V2) where the orientations of the Guillotine cuts are fixed.

As positive results, we present pseudo-polynomial time algorithms for both problem variants (V3) and (V4), and then turn them into a fully polynomial time approximation scheme (FPTAS, for short). A standard way of dealing with NP-hard problems is not to search for an optimal solution, but to go for *near-optimal* solutions. An algorithm that returns near-optimal solutions with cost at most a factor ρ above the optimal cost (where $\rho > 1$ is some fixed real number) is called a *ρ-approximation algorithm*. A *fully polynomial time approximation scheme* (FPTAS) is a family of $(1+\varepsilon)$-approximation algorithms over all $\varepsilon > 0$ with running time polynomially bounded in the input size and in $1/\varepsilon$ (see also [2]). With respect to relative performance guarantees, an FPTAS is the strongest possible polynomial time approximation result that we can derive for an NP-hard problem. Hence, our results demonstrate that both problem variants (V3) and (V4) behave very well with respect to polynomial time approximation algorithms.

Organization of this paper. In Section 2 we will show that both variants (V3) and (V4) are NP-complete, in Section 3 we present pseudo-polynomial time algorithms for them, and in Section 4 we construct FPTASs for them. Section 5 completes the paper with a brief conclusion.

2. The NP-completeness proof

In this section we prove that the problem variants (V3) and (V4) both are NP-complete in the ordinary sense. Both problems clearly are contained in the complexity class NP, and so we only need to establish their NP-hardness. The NP-hardness proof is done by a reduction from the PARTITION problem that is known to be NP-complete in the ordinary sense; see [2].

Problem: PARTITION.

Instance: A sequence $q_1, \ldots, q_n$ of positive integers such that $\sum_{i=1}^{n} q_i = 2Q$.

Question: Does there exist a partition of the numbers q_i into two groups such that the elements of each group add up to exactly Q?

Now consider an arbitrary instance I of the PARTITION problem. Without loss of generality we assume that $q_i \leqslant Q$ holds for $i = 1, \ldots, n$.

We construct the following slicing tree T_I from I: The backbone of the tree T_I consists of a chain of $n+1$ interior vertices $k_0, \ldots, k_n$. The first vertex k_0 forms the root of T_I. For $i = 0, \ldots, n-1$, the interior vertex k_i has the interior vertex k_{i+1} as a right child. The remaining $n+2$ vertices in T_I are the leaves $\ell_0, \ldots, \ell_{n+1}$. The vertex k_i $(0 \leqslant i \leqslant n)$ has leaf

ℓ_i as a left child. Moreover, k_n has leaf ℓ_{n+1} as a right child. The cell corresponding to leaf ℓ_i ($1 \leqslant i \leqslant n$) has width and height parameters $a = b = q_i$. The cell corresponding to leaf ℓ_{n+1} has parameters $a = b = Q$, and the cell corresponding to leaf ℓ_0 has parameters $a = b = 2Q$. This completes the description of the slicing tree T_I. Since in T_I all the VLSI components associated with the leaves are squares, the orientations of cells become irrelevant. Therefore, the problem variants (V3) and (V4) coincide for such an input tree T_I, and our reduction works for both variants.

Lemma 2.1. *There exists a layout in $\mathscr{F}(T_I)$ with area at most $8Q^2$, if and only if the instance I of PARTITION has answer YES.*

Proof. Let us first consider the possible floorplan layouts for the subtree T' that is rooted at vertex k_1, the interior vertex that is just below the root: These layouts start from the basic $Q \times Q$ cell associated with the leaf ℓ_{n+1}. While moving upwards through the tree, the $q_i \times q_i$ cells for leaf ℓ_i are added one by one to this basic cell. If the interior vertex k_i is fixed as a vertical Guillotine cut, then this adds q_i to the width of the floorplan but leaves its height unchanged (since we assume $q_i \leqslant Q$). And if the interior vertex k_i is fixed as a horizontal Guillotine cut, then this adds q_i to the height of the floorplan but leaves its width unchanged. For such a floorplan layout for the subtree T', we denote by $J \subseteq \{1, \ldots, n\}$ the set of indices i for which vertex k_i is fixed as a vertical Guillotine cut. Then this floorplan layout has width equal to $Q + \sum_{i \in J} q_i$, and it has height equal to $Q + \sum_{i \notin J} q_i$. Since $\sum_{i \notin J} q_i = 2Q - \sum_{i \in J} q_i$, the width and height are of the form $Q + x$ and $3Q - x$ with $x = \sum_{i \in J} q_i$.

Now let us prove the statement in the lemma. If the instance I of PARTITION has answer YES, then there exists a subset $J \subseteq \{1, \ldots, n\}$ such that $x = \sum_{i \in J} q_i = Q$. For this J and x, the above floorplan for T' forms a square with side length $2Q$. This $2Q \times 2Q$ square can be combined with the $2Q \times 2Q$ square in leaf ℓ_0 by either a vertical or a horizontal subdivision in k_0. The resulting layout for T_I is a $4Q \times 2Q$ rectangle of area $8Q^2$.

If the instance I of PARTITION has answer NO, then for any subset $J \subseteq \{1, \ldots, n\}$ we have $x = \sum_{i \in J} q_i \neq Q$ with $0 \leqslant x \leqslant 2Q$. Consider a floorplan layout for T' with width $Q + x$ and with height $3Q - x$. We distinguish two cases. In the first case k_0 is a vertical subdivision, and the resulting floorplan for T_I has width $2Q + (Q + x) = 3Q + x$ and height $\max\{2Q, 3Q - x\}$. The resulting area is

$$(3Q + x)\max\{2Q, 3Q - x\} = \max\{6Q^2 + 2Qx, 9Q^2 - x^2\} > 8Q^2. \quad (1)$$

Here the inequality follows, since for $x < Q$ we have $9Q^2 - x^2 > 8Q^2$, and for $x > Q$ we have $6Q^2 + 2Qx > 8Q^2$, and $x = Q$ is impossible. In the second case k_0 is a horizontal subdivision, and the resulting floorplan for T_I has width $\max\{2Q, Q + x\}$ and height $2Q + (3Q - x) = 5Q - x$. The resulting area is

$$(5Q - x)\max\{2Q, Q + x\} = \max\{10Q^2 - 2Qx, 5Q^2 + 4Qx - x^2\} > 8Q^2. \quad (2)$$

Here the inequality follows, since for $x < Q$ we have $10Q^2 - 2Qx > 8Q^2$, and for $Q < x \leqslant 2Q$ we have $5Q^2 + 4Qx - x^2 > 8Q^2$, and $x = Q$ is impossible. This completes the proof of the lemma. □

As an immediate consequence of Lemma 2.1 and of the fact that PARTITION is NP-complete in the ordinary sense, we get the following theorem.

Theorem 2.2. *The variants (V3) and (V4) of slicing floorplan designs both are NP-complete in the ordinary sense.*

Note that the above constructed slicing tree T_I is totally unbalanced and has depth $\Theta(n)$. What about balanced trees? In this case the problem becomes much easier: one of the results in Almeida et al. [1] yields that the optimal floorplan for a slicing tree of depth d can be found in $O(d4^d)$ time. A perfectly balanced binary tree with n leaves has depth $\log n$, and hence by [1] the optimal floorplan can be computed in polynomial time $O(n^2 \log n)$.

3. The pseudo-polynomial time algorithm

In this section we derive a pseudo-polynomial time algorithm for the two layout problems (V3) and (V4) that is based on a dynamic programming approach.

Let T be an arbitrary slicing tree. For a vertex $v \in T$, we denote by $T(v)$ the induced subtree of T that consists of vertex v and all the vertices below v. Hence, for the root r of T we have $T(r) = T$. Let L denote the sum of all parameter values a and all parameter values b in the leaves of T. Clearly, L is pseudo-polynomially bounded in the input size. Moreover, any reasonable floorplan for T will have width and height at most L, and from now on we will restrict our attention to such floorplans.

Definition 3.1. For a vertex $v \in T$, we denote by $S(v)$ the set of all pairs (w,h) of integers with $1 \leqslant w \leqslant L$ and $1 \leqslant h \leqslant L$ such that there exists a floorplan layout for the slicing tree $T(v)$ of width w and of height h.

Our first goal is to determine all the sets $S(v)$ with $v \in T$. This is done by moving bottom-up through the tree T, starting in the leaves and ending in the root. Whenever a vertex is handled, its children already will have been handled. In the initialization phase, we handle the leaves: for a leaf v with width parameter a and height parameter b, we have $S(v) = \{(w,h) : w \geqslant a, h \geqslant b\}$ in variant (V3), and we have

$$S(v) = \{(w,h) : w \geqslant a, h \geqslant b\} \cup \{(w,h) : w \geqslant b, h \geqslant a\} \quad (3)$$

in variant (V4). The remaining steps in the dynamic program do not depend on rotations of components or cells, and hence will be identical for both variants (V3) and (V4).

Now let us turn to an interior vertex v with left child v_ℓ and right child v_r. We simply combine all possible floorplans described by the set $S(v_\ell)$ with all possible floorplans described by the set $S(v_r)$. These combinations depend on whether v is fixed as a vertical or as a horizontal Guillotine cut. If v is fixed vertically, then the widths of the two layouts for $T(v_\ell)$ and $T(v_r)$ simply add up, and the new height is the maximum of the heights of the two layouts. If v is fixed horizontally, we are in a symmetric situation with the roles of width and height exchanged. Summarizing, this discussion yields

$$\begin{aligned} S(v) = \{&(w_\ell + w_r, \max\{h_\ell, h_r\}) : (w_\ell, h_\ell) \in S(v_\ell), \\ &(w_r, h_r) \in S(v_r)\} \cup \{(\max\{w_\ell, w_r\}, h_\ell + h_r) : \\ &(w_\ell, h_\ell) \in S(v_\ell), (w_r, h_r) \in S(v_r)\}. \end{aligned}$$

This completes the description of the computation of all sets $S(v)$ with $v \in T$. In order to find the smallest possible floorplan area, we simply search through the set $S(r)$ associated with the root r to T, and we output the minimium value wh with $(w,h) \in S(r)$.

Since $S(v) \subseteq \{1,\ldots,L\} \times \{1,\ldots,L\}$ for all $v \in T$, we have $|S(v)| \leqslant L^2$. Hence, every set $S(v)$ can be determined in $O(L^4)$ time, and the overall running time is $O(nL^4)$. Similarly as in [7], we can speed-up this running time by disregarding the *dominated* floorplans from the sets $S(v)$. We say that a floorplan with dimensions (w_1,h_1) is *dominated by another floorplan with dimensions* (w_2,h_2), if $w_1 \geqslant w_2$ and $h_1 \geqslant h_2$ hold. In other words, a dominated floorplan in both dimensions is no better than the dominating floorplan. Clearly, if a floorplan in $S(v)$ is dominated by another floorplan in $S(v)$, then it can be removed without losing anything towards the optimal solution.

This suggests the following modified dynamic program. Whenever a set $S(v)$ has been determined, then we clean it up and remove all dominated solutions from it. The resulting set of undominated floorplans is called $\mathrm{US}(v)$. All further computations are then done with this set $\mathrm{US}(v)$ instead of set $S(v)$. Since $\mathrm{US}(v)$ contains at most one pair (w,h) for every fixed value h with $1 \leqslant h \leqslant L$, we have $|\mathrm{US}(v)| \leqslant L$. The computation of $S(v)$ takes only $O(L^2)$ time, and also the cleaning up for $\mathrm{US}(v)$ can be done in $O(L^2)$ time. Hence, the overall running time of this modified approach is $O(nL^2)$.

Theorem 3.2. *The variants* (*V*3) *and* (*V*4) *of slicing floorplan designs can be solved in pseudo-polynomial time* $O(nL^2)$, *where n denotes the number of leaves and L denotes the sum of all parameter values a and all values b in the leaves of T.*

The described algorithm computes the optimal area. By storing appropriate auxiliary information in the states of the dynamic program, one can compute the structure of the corresponding optimal layout within the same asymptotic time bounds. Since these are standard techniques, we do not elaborate on them.

4. The fully polynomial time approximation scheme

In this section we design a FPTAS for the two layout problems (V3) and (V4). For doing this, we will build on the pseudo-polynomial time algorithm in Section 3, and transform it into an FPTAS. This is done via the so-called approach of trimming the state space of the dynamic program, a standard approach in the area (see for instance [3,9]). The main idea is to iteratively thin out the state space of the dynamic program, to collapse solutions that are 'close' to each other, and to bring the size of the state space down to polynomial.

Let T be an arbitrary slicing tree with n leaves. Exactly as in the preceding Section 3, let L denote the sum of all parameter values a and all parameter values b in the leaves of T. Let $\varepsilon > 0$ be a small real number (that will be precision of approximation). Next, we introduce several concepts that we will use in constructing the FPTAS. We define the so-called *trimming* parameter Δ as

$$\Delta = 1 + \frac{\varepsilon}{8n}. \tag{4}$$

Furthermore, we define

$$k = \lceil \log_{\Delta}(L) \rceil, \tag{5}$$

where $\log_{\Delta}(\cdot)$ denotes the base Δ logarithm. We partition the interval $[0, L]$ into k intervals $\mathscr{I}_1, \ldots, \mathscr{I}_k$. For $i = 1, \ldots, k-1$ there is the half-open interval $\mathscr{I}_i = [\Delta^{i-1}, \Delta^i)$. Moreover, there is the closed interval $\mathscr{I}_k = [\Delta^{k-1}, L]$. Note that every integer in the range $[0, L]$ is contained in precisely one of these intervals. Note furthermore that the left and the right endpoint of each interval are at most a factor of Δ away from each other. Finally, we define a partition of the integer points in $\{1, \ldots, L\} \times \{1, \ldots, L\}$ into k^2 orthogonal, axes-parallel boxes: every such box is the product of some interval $\mathscr{I}_i$ in the width coordinate with some interval $\mathscr{I}_j$ in the height coordinate. These boxes will be called the *Δ-boxes*.

Now let us turn to the FPTAS. We will essentially follow the dynamic programming algorithm from Section 3, but we will modify and shrink the sets $S(v)$ such that their cardinalities become polynomial in the input size. We will not care about dominated and undominated solutions, but we will simply follow the very primitive first approach where the sets $S(v)$ encode all feasible solutions for the tree $T(v)$. Whenever a set $S(v)$ has been determined for a vertex v, then we clean it up in the following way and produce a so-called *trimmed* set $S^{\#}(v)$: from every Δ-box $\mathscr{B}$, we keep at most one (arbitrary) solution for $S^{\#}(v)$. More precisely, $S(v) \cap \mathscr{B} \neq \emptyset$ holds if and only if $|S^{\#}(v) \cap \mathscr{B}| = 1$. All further computations are then done with this set $S^{\#}(v)$ instead of set $S(v)$. The intuition for this lies in the fact that in a Δ-box, the width-coordinates of all points are at most a factor Δ away from each other, and also their height-coordinates are at most a factor Δ away from each other. Since Δ is very close to one, all the points in $\mathscr{B}$ are fairly close to each other. Hence, we do not lose a lot if we throw away most of these points, and only keep one of them as a representative of the whole Δ-box $\mathscr{B}$.

In the rest of this section, we will make this intuition mathematically precise. We will show that the area of the delivered solution is at most a factor of $1 + \varepsilon$ above the area of the optimal solution for the instance T, and that the modified dynamic program has a polynomial running time.

Lemma 4.1. *The area of the solution found by the modified dynamic program is at most a factor of $1 + \varepsilon$ above the (optimal) area found by the original dynamic program.*

Proof. Whenever we clean up a solution set and generate a set $S^{\#}(v)$, we introduce a multiplicative error of at most Δ in the width-coordinate, and a multiplicative error of at most Δ in the height-coordinate. This yields a multiplicative error of at most Δ^2 in the area. Since a binary tree with n leaves has exactly $n - 1$ interior vertices, we gen-

erate less than $2n$ trimmed solution sets. Hence, the overall introduced error is bounded by

$$\Delta^{4n} = \left(1 + \frac{\varepsilon}{8n}\right)^{4n} \leqslant 1 + \varepsilon. \tag{6}$$

Here we have used that for any $0 \leqslant x \leqslant 1$ and for any $m \geqslant 1$, the inequality $(1 + x/m)^m \leqslant 1 + 2x$ holds. This can be seen as follows: the left-hand side $(1 + x/m)^m$ is a convex function in x, and the right-hand side $1 + 2x$ is a linear function in x. Moreover, the claimed inequality holds at $x = 0$ and $x = 1$. This completes the proof of the lemma. □

Lemma 4.2. *The running time of the modified dynamic program is polynomially bounded in* n, *in* $\ln(L)$, *and in* $1/\varepsilon$.

Proof. First observe that there are only k^2 different Δ-boxes, and that every trimmed set $S^{\#}(v)$ contains at most one point from every box. Therefore, $|S^{\#}(v)| \leqslant k^2$ holds for every $v \in T$. Whenever the solution sets for the children v_ℓ and v_r are combined to give the solution set for vertex v, this costs $O(|S^{\#}(v_\ell)||S^{\#}(v_r)|)$ time. As a consequence, every trimmed set $S^{\#}(v)$ can be determined in $O(k^4)$ time, and the overall running time of the modified dynamic program is $O(k^4 n)$.

Now let us get an upper bound on the value $k = \lceil \log_\Delta (L) \rceil$ where Δ is defined as in (4). We use the well-known inequality $\ln x \geqslant (x-1)/x$ for $x \geqslant 1$ (this inequality can be easily derived from the Taylor expansion of $\ln x$). We get

$$k = \left\lceil \frac{\ln(L)}{\ln(\Delta)} \right\rceil \leqslant \left\lceil \ln(L) \frac{\Delta}{\Delta - 1} \right\rceil = \left\lceil \ln(L)\left(1 + \frac{8n}{\varepsilon}\right) \right\rceil. \tag{7}$$

Hence, k is polynomially bounded in n, $\ln(L)$, and $1/\varepsilon$, and so is the running time of the modified dynamic program. □

Note that the number of bits in the specification of the input is at least $\log_2(L)$. With this, Lemma 4.2 states that the running time of the modified dynamic program is polynomially bounded in the input size n and $\log_2(L)$, and in the reciprocal value of the precision ε. That is all we need for an FPTAS.

Theorem 4.3. *The variants* (*V*3) *and* (*V*4) *of slicing floorplan designs both possess a FPTAS.*

5. Conclusion

In this paper, we have performed a complete analysis of the complexity and the approximability of two slicing floorplan layout problems. These problems are NP-hard in the ordinary sense, they are solvable in pseudo-polynomial time, and they possess a FPTAS.

Acknowledgement

Supported by the START program Y43-MAT of the Austrian Ministry of Science.

References

[1] A.M.C. Almeida, E.Q.V. Martins, R.D. Rodrigues, Optimal cutting directions and rectangle orientation algorithm, European Journal of Operational Research 109 (1998) 660–671.
[2] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.
[3] O. Ibarra, C.E. Kim, Fast approximation algorithms for the knapsack and sum of subset problems, Journal of the ACM 22 (1975) 463–468.
[4] U. Lauther, A min-cut placement algorithm for general cell assemblies based on a graph representation, Journal of Digital Systems 4 (1980) 21–34.
[5] R.H.J.M. Otten, Automatic floorplan design, in: Proceedings of the 19th ACM IEEE Design and Automation Conference, 1982, pp. 261–267.
[6] R.H.J.M. Otten, What is a floorplan? in: Proceedings of the International Symposium on Physical Design (ISPD'00), 2000, pp. 201–206.
[7] L. Stockmeyer, Optimal orientations of cells in slicing floorplan designs, Information and Control 57 (1983) 91–101.
[8] T. Wang, D.F. Wong, Optimal floorplan area optimization, IEEE Transactions on Computer-Aided Design 11 (1992) 992–1002.
[9] G.J. Woeginger, When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? INFORMS Journal on Computing 12 (2000) 57–75.