An Optimal Algorithm for Area Minimization of Slicing Floorplans

Weiping Shi*
Department of Computer Science
University of North Texas
Denton, Texas 76203

Abstract

The traditional algorithm of Stockmeyer for area minimization of slicing floorplans has time (and space) complexity $O(n^2)$ in the worst case, or $O(n \log n)$ for balanced slicing. For more than a decade, it is considered the best possible.

In this paper, we present a new algorithm of worst-case time (and space) complexity $O(n \log n)$, where n is the total number of realizations for the basic blocks, regardless whether the slicing is balanced or not. We also prove $\Omega(n \log n)$ is the lower bound on the time complexity of any area minimization algorithm. Therefore, the new algorithm not only finds the optimal realization, but also has an optimal running time.

1 Introduction

1.1 Problem Description

A *floorplan* F is a dissection of a rectangle into a set of non-intersecting basic blocks $B_1, B_2, \ldots, B_m$, see Figure 1. A floorplan also associates each basic block B_i with a set of cells $C_i \subset \mathbf{R}^+ \times \mathbf{R}^+$, where $\mathbf{R}^+$ is the set of positive real numbers. C_i specifies the widths and heights of rectangle cells that must fit in B_i. If we select a cell $c_{j_i} \in C_i$ for each block B_i, $i = 1, 2, \ldots, m$, and arrange them according to the floorplan F, then we obtain a *realization* ρ of F. Following this definition, cells in C_i are also called realizations of the basic block B_i. Given a floorplan F and a realization ρ, there are real numbers $h(F, \rho)$ and $w(F, \rho)$ giving the minimum height and width, respectively, of a floorplan that is equivalent to F for which each cell c_{j_i} of ρ fits in B_i. The *area minimization* (area optimization) problem is to find a realization ρ of F such that the area, $h(F, \rho) \times w(F, \rho)$, is minimum among all realizations.

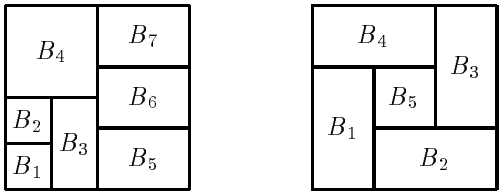

Figure 1: A slicing floorplan (a) and a non-slicing floorplan (b).

Floorplans are classified as slicing or non-slicing. A floorplan is *slicing* if either it is a basic block, or there is a line segment that divides the enclosing rectangle into two sub-floorplans such that each sub-floorplan is slicing, see Figure 1(a). A floorplan is *non-slicing* if it is not a slicing floorplan. Figure 1(b) is an example non-slicing floorplan called a *wheel*. A slicing floorplan is represented by a rooted binary tree called a *slicing tree*, see Figure 2. Each non-leaf node in the tree is labeled either h or v, specifying whether it is a horizontal or vertical slice. Each leaf node corresponds to a basic block. For every internal node in the slicing tree, the sub-tree with that node being the root defines a sub-floorplan.

Floorplans are also classified as hierarchical or non-hierarchical. A *hierarchical* floorplan is one that can be constructed recursively by a pattern of fixed size, such as a vertical slice, a horizontal slice, or a wheel. Otherwise, the floorplan is *non-hierarchical*.

1.2 Review of Previous Work

Many researchers have studied various aspects of area minimization of floorplans. For slicing floorplans, Otten [6] first showed the minimization problem can be solved efficiently. Stockmeyer [10] in 1983 presented

*Supported in part by NSF grant MIP-9309120.

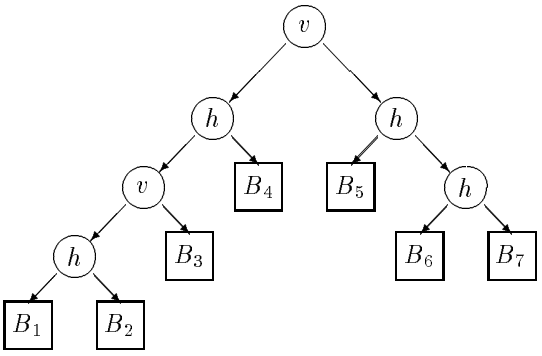

Figure 2: The slicing tree for the slicing floorplan of Figure 1(a).

the now widely used algorithm of time complexity $O(nd)$, where n is the number of basic blocks, d is the depth of the slicing tree, and each basic block has two realizations. Thus, the time is $O(n^2)$ in the worst case, or $O(n \log n)$ for balanced slicing with $d = O(\log n)$. The worst-case time complexity is indeed $\Theta(n^2)$ when the depth of the slicing tree is $\Theta(n)$. The space complexity is the same as the time complexity. Although one can minimize the depth of some part of the slicing tree if that part contains only vertical slices or only horizontal slices, many trees cannot be balanced.

For hierarchical non-slicing floorplans, Pan, Shi and Liu [8] recently proved the problem is weakly NP-complete. For the non-slicing floorplan of Figure 1(b), where each of the five basic blocks has k realizations, Wang and Wong [13] proposed an $O(k^3 \log k)$ time algorithm, which was improved by Chen and Tollis [2], as well as Pan, Shi and Liu [8] to $O(k^2 \log k)$. For hierarchical non-slicing floorplans, algorithms were developed by Wang and Wong [13], Chen and Tollis [2], and pseudo-polynomial time algorithms were developed by Pan, Shi and Liu [8].

For non-hierarchical floorplans, Stockmeyer [10] proved the problem is strongly NP-complete. Wimer, Koren and Cedernaumn [14], as well as Chong and Sahni [3] proposed branch-and-bound algorithms. Pan and Liu [7] developed algorithms for general floorplans that are "approximately" slicing.

Finally, we review other related research. Sarrafzadeh [9] investigated how to transform an arbitrary floorplan into a slicing one, by slightly increasing the area. His result makes algorithms for slicing floorplans also applicable to non-slicing floorplans. Dai and Kuh [4], Zimmermann [15], etc, studied how to combine floorplanning and wiring to achieve good overall performance. For more information, see Lengauer [5].

2 New Algorithm

For any two realizations ρ_1 and ρ_2 of floorplan F, ρ_1 is said to *dominate* ρ_2 if $w(F, \rho_1) \leq w(F, \rho_2)$ and $h(F, \rho_1) \leq h(F, \rho_2)$. Intuitively, ρ_1 dominates ρ_2 if ρ_1 is smaller than ρ_2 in both height and width. Since our objective is to minimize the area, it is obvious that we do not need ρ_2. This is true when F is either the floorplan, or a sub-floorplan, or a basic block. For any floorplan F, the set of *nonredundant realizations* of F, $R(F)$, is a set of realizations of F such that 1) no realization in $R(F)$ is dominated by any other realization of F, and 2) every realization of F must be dominated by a realization in $R(F)$. It is obvious that for any floorplan F, we only need to keep $R(F)$.

The new algorithm works as follows. Suppose we are given a floorplan F which consists of two sub-floorplans F_1 and F_2 sliced vertically or horizontally. We first recursively compute the two sets of nonredundant realizations $R(F_1)$ and $R(F_2)$, and store $R(F_1)$ and $R(F_2)$ in a data structure called realization trees (Section 2.1). Then we use a new merging algorithm (Section 2.2) to combine $R(F_1)$ and $R(F_2)$ to get $R(F)$ and store $R(F)$ in a realization tree.

2.1 Data Structure

We assume the readers have some familiarity with the properties and uses of balanced search trees such as AVL trees [11]. In general, a balanced binary search tree allows time $O(\log n)$ for search, insertion, and deletion of any key, where n is the number of vertices in the tree. In this paper, we need an additional requirement that the search, insertion, and deletion of any key must be performed in time $O(\log k)$, where k is the number of vertices between the position we start to search and the position we find the key. This additional property, known as finger search, is satisfied by AVL trees [12].

It is not hard to prove if $\sum_{i=1}^{n_2} k_i \leq n_1$, then

$$\sum_{i=1}^{n_2} O(\log k_i) = O\left(n_2 \log\left(1 + \frac{n_1}{n_2}\right)\right). \qquad (1)$$

This formula implies that given a balanced binary search tree of n_1 vertices satisfying the finger search requirement, then the search, insertion and deletion of n_2 keys in sorted order can be performed in total time $O(n_2 \log(1 + \frac{n_1}{n_2}))$.

For any floorplan F, which may be the whole floorplan, or a sub-floorplan, or a basic block, we use a *realization tree* $T(F)$ to store $R(F)$, the set of nonredundant realizations of F. $T(F)$ is organized as an

AVL tree. For every realization $\rho \in R(F)$, there is a unique vertex $v(\rho) \in T(F)$. Every vertex $v \in T(F)$ is associated with the following fields related to the key:

$h(v)$	:	to be used to compute height;
$w(v)$	:	to be used to compute width;
$h^+(v)$	:	to be added to heights of all descendents;
$w^+(v)$	:	to be added to widths of all descendents.

In fact, $T(F)$ has two keys, the height and the width of the realizations. $T(F)$ is organized in increasing height order, and also in decreasing width order. This is possible only because the realizations are nonredundant. The search, insertion and deletion can be performed under either key. However, the height and width of a realization ρ are not stored explicitly in the corresponding vertex $v(\rho)$. Instead, the information is stored along the path from the root of $T(F)$ to $v(\rho)$. Let $P(\rho)$ be the path from the root to $v(\rho)$, including the root and $v(\rho)$, then

$$\begin{aligned} h(F,\rho) &= h(v(\rho)) + \sum_{u \in P(\rho)} h^+(u), \\ w(F,\rho) &= w(v(\rho)) + \sum_{u \in P(\rho)} w^+(u). \end{aligned}$$

To remember the composition of realization ρ, the vertex $v(\rho)$ also has two fields $p(v)$ and $p^+(v)$. In addition, each vertex $v(\rho) \in T(F)$ has the standard fields such as $left(v)$, $right(v)$ and $balance(v)$. Field $left(v)$ points to a sub-tree containing realizations of heights less than (and widths greater than) the height (width) of ρ. Field $right(v)$ points to a sub-tree containing realizations of heights greater than (and widths less than) the height (width) of ρ. Field $balance(v)$ contains information for re-balancing the tree which we will not discuss in this paper. Figure 3 is an example realization tree of five realizations.

The search of realization trees is similar to the search of ordinary AVL trees, except the values of $h^+(v)$ and $w^+(v)$ are added to $h(v)$ and $w(v)$, and propagated to the two children of v whenever v is visited. The reason for the "lazy" propagation is to avoid repeated unnecessary updates.

2.2 Merge of Floorplans

Suppose we are given a floorplan F which consists of two sub-floorplans F_1 and F_2 sliced vertically. Assume recursively $R(F_1)$ and $R(F_2)$ have been computed and stored in realization trees $T(F_1)$ and $T(F_2)$ respectively. Let the number of nodes in $T(F_1)$ and $T(F_2)$ be n_1 and n_2, and assume without loss of generality

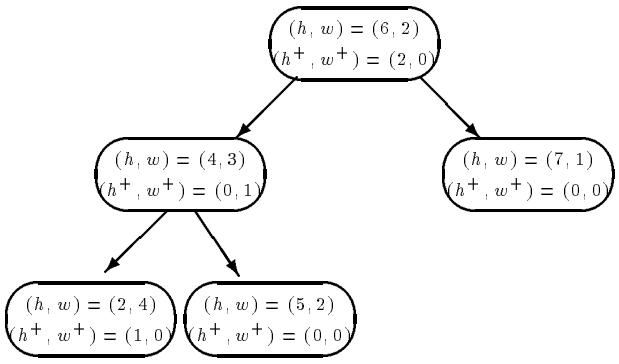

Figure 3: A realization tree containing five realizations: (5,5), (6,4), (7,3), (8,2) and (9,1).

$n_1 \geq n_2$. We show how to construct $R(F)$ and store it in $T(F)$.

The first step of the merging algorithm deals with nonredundant realizations of F such that the heights are decided by F_2. For each realization $\rho_i \in R(F_2)$, we want to find a realization $\sigma_j \in R(F_1)$ such that the height of σ_j is less than the height of ρ_i and the width of σ_j is the minimum among all such σ_j's. Given ρ_i, we can find such σ_j by searching $T(F_1)$. Together, ρ_i and σ_j form a realization of F with height being $h(F_2, \rho_i)$ and width being $w(F_2, \rho_i) + w(F_1, \sigma_j)$. The composition of the realization is represented by P which points to $p(v(\rho_i))$ and $p(v(\sigma_j))$.

So we enumerate $R(F_2)$ in increasing height order, and for every $\rho_i \in R(F_2)$, we search $T(F_1)$ for the corresponding σ_j. Since ρ_i's are in increasing height order, σ_j's must be in non-decreasing height order. Therefore, the total time to enumerate $R(F_2)$ is $O(n_2)$ and the total time to search $T(F_1)$ is $O(n_2 \log(1 + \frac{n_1}{n_2}))$. The newly generated realizations are stored in a temporary list in increasing height order for later use. The size of this list is at most n_2.

The second step of the merging algorithm deals with nonredundant realizations of F such that the heights are decided by F_1. We also include cases where the heights are decided by both F_1 and F_2 simultaneously. For each realization $\rho_i \in R(F_2)$, we want to find realizations $\sigma_j, \sigma_{j+1}, \ldots, \sigma_l$ in $R(F_1)$ such that

$$\begin{aligned} j &= \min_{1 \leq k \leq n_1} \{k \mid \sigma_k \in R(F_1), h(F_1, \sigma_k) \geq h(F_2, \rho_i)\}, \\ l &= \max_{1 \leq k \leq n_1} \{k \mid \sigma_k \in R(F_1), h(F_1, \sigma_k) < h(F_2, \rho_{i+1})\}. \end{aligned}$$

This can be done through two searches of $T(F_1)$ using the heights of ρ_i and ρ_{i+1}. If no such j and l are found, then we repeat the above for the next realization $\rho_{i+1} \in R(F_2)$. Otherwise, we can form the

following $l-j+1$ realizations of F:

$$\begin{aligned}
\rho_i \text{ and } \sigma_j &: (h(F_1,\sigma_j), w(F_1,\sigma_j)+w(F_2,\rho_i)),\\
\rho_i \text{ and } \sigma_{j+1} &: (h(F_1,\sigma_{j+1}), w(F_1,\sigma_{j+1})+w(F_2,\rho_i)),\\
&\ldots\\
\rho_i \text{ and } \sigma_l &: (h(F_1,\sigma_l), w(F_1,\sigma_l)+w(F_2,\rho_i)).
\end{aligned}$$

To record the newly generated realizations, we change the keys of vertices $v(\sigma_j),\ldots,v(\sigma_l)$ in $T(F_1)$. This way, step by step, $T(F_1)$ is changed into a tree containing realizations of F. However, we cannot afford spending $O(l-j)$ time to change the vertices one by one since otherwise we will have the same time complexity as Stockmeyer's. Instead, we use the fields h^+ and w^+. Figure 4 illustrates the general situation of the vertices $v(\sigma_j)$, $v(\sigma_{j+1})$, ..., $v(\sigma_l)$.

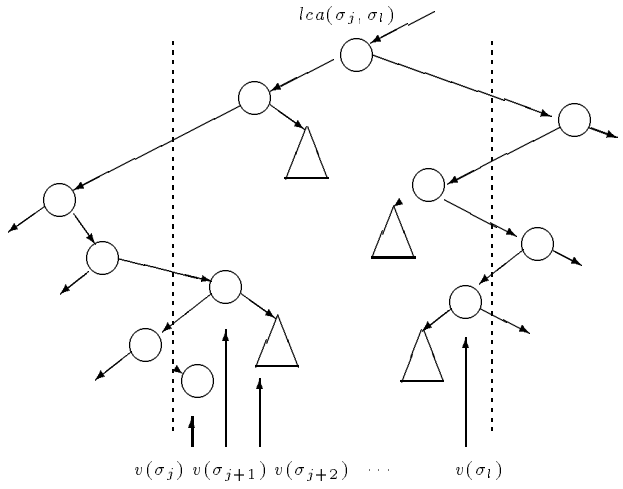

Figure 4: Vertices $v(\sigma_j), v(\sigma_{j+1}),\ldots,v(\sigma_l)$ form an interval in the realization tree.

Let $lca(\sigma_j,\sigma_l)$ be the *least common ancestor* of $v(\sigma_j)$ and $v(\sigma_l)$, see Figure 4. Let the *left boundary* be the set of vertices u such that u is on the path from $v(\sigma_j)$ to $lca(\sigma_j,\sigma_l)$ and the height (width) of u is greater (less) than or equal to the height (width) of σ_j. For every left boundary vertex u, including $v(\sigma_j)$ but not $lca(\sigma_j,\sigma_l)$, we make the following changes to its fields:

$$\begin{aligned}
w(u) &\leftarrow w(u)+w(F_2,\rho_i),\\
w^+(right(u)) &\leftarrow w^+(right(u))+w(F_2,\rho_i).
\end{aligned}$$

Similarly, we can define *right boundary* and make changes to the fields accordingly. For vertex $lca(\sigma_j,\sigma_l)$, we make the following change:

$$w(lca(\sigma_j,\sigma_l)) \leftarrow w(lca(\sigma_j,\sigma_l))+w(F_2,\rho_i).$$

It can be shown that the number of vertices in the left and right boundaries is $O(\log(l-j))$. From (1), this step can be done in total time $O(n_2\log(1+\frac{n_1}{n_2}))$.

In the third and also the final step, we insert the temporary list of size n_2 generated in the first step, into the realization tree of size n_1 obtained in the second step. We also delete redundant realizations during the merge. Again from the finger search property, this step can be done in total time $O(n_2\log(1+\frac{n_1}{n_2}))$. When we finish, the tree becomes $T(F)$, the realization tree for $R(F)$.

If the floorplan consists of two sub-floorplans sliced horizontally, then exchange the words "height" with "width", "h" with "w", etc, to the above three steps and everything should follow.

As the basis of the recursion, if the floorplan F is a basic block B_i, then we create a realization tree $T(B_i)$, where every vertex of $T(B_i)$ represents one non-redundant realization in C_i. In other words, if there is a cell $c_j \in C_i$, then we have a vertex $v_j \in T(B_i)$, such that $w(v_j)=w(c_j)$, $h(v_j)=h(c_j)$, $w^+(v_j)=h^+(v_j)=0$. This can be done in time $O(|C_i|\log|C_i|)$.

2.3 Time and Space Complexity

Let $\mathcal{T}(n)$ be the worst-case time complexity of the new algorithm for floorplan F that contains n realizations for basic blocks. Keep in mind the fact that for any slicing floorplan F containing n realizations of basic blocks, there are at most n nonredundant realizations of F. Therefore, we have the following recurrence relation:

$$\mathcal{T}(n) \le \begin{cases} c_1 n\log n & F \text{ is basic block,}\\ \max\{\mathcal{T}(n_1)+\mathcal{T}(n_2)+ & \\ \quad +c_2 n_2\log\left(1+\frac{n_1}{n_2}\right)\} & \text{otherwise}\end{cases}$$

where c_1, c_2 are constants, n_1, n_2 are the number of realizations of basic blocks of F_1 and F_2 respectively, and the maximum is taken over all n_1, n_2 such that $n_1+n_2=n$ and $n>n_1\ge n_2>0$. Using induction, we can prove that $\mathcal{T}(n)=O(n\log n)$.

The space complexity is bounded by the time complexity which is $O(n\log n)$. However, if we just compute as output the minimum area instead of the composition of the realization, then we can reduce the space complexity to $O(n)$ by avoiding the fields $p(v)$ and $p^+(v)$.

3 Lower Bound

Ben-Or [1] proved the following problem requires $\Omega(n \log n)$ algebraic operations, where algebraic operations include $+, -, *, /, \sqrt{}, =, >, \geq$, etc.

> *Set Disjointness Problem.* Given two sets of positive real numbers $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$, determine whether there are indices i and j such that $x_i = y_j$.

We show a reduction from the set disjointness problem to the area minimization of slicing floorplan problems. Given an instance of the set disjointness problem X and Y, construct a floorplan with only two basic blocks B_1 and B_2 sliced vertically, and associate the following cells to B_1 and B_2:

$$C_1 = \{(x_1, 1/x_1), (x_2, 1/x_2), \ldots, (x_n, 1/x_n)\},$$
$$C_2 = \{(y_1, 1/y_1), (y_2, 1/y_2), \ldots, (y_n, 1/y_n)\}.$$

Then it is easy to see the floorplan has minimum area 2 iff there are indices i and j such that $x_i = y_j$. Since the set disjointness problem requires $\Omega(n \log n)$ operations, the area minimization problem also requires $\Omega(n \log n)$ operations.

4 Conclusion and Simulation

We presented a new algorithm of worst-case time complexity $O(n \log n)$ and space complexity $O(n \log n)$, regardless whether the slicing tree is balanced, or the number of realizations for the basic blocks is balanced. This is an improvement, both in time an in space, of Stockmeyer's $O(n^2)$-time $O(n^2)$-space algorithm [10]. We also proved $\Omega(n \log n)$ is the lower bound on the time complexity of any area minimization algorithm even if there are only two basic blocks.

Both the new algorithm and Stockmeyer's algorithm are implemented in C on a Sequent Symmetry computer. Preliminary results indicate for balanced floorplans, the new algorithm is about twice slower for all values of n due to the data structure overhead. But for unbalanced floorplans, the new algorithm is much faster for $n > 200$, see Table 1. The new algorithm performs best when the slicing tree is unbalanced, or when the number of realizations for basic blocks are unbalanced.

Acknowledgment The author thanks Larry Stockmeyer for constructive comments on an earlier version of the paper, Steve Tate for discussions, and Hongfeng Li for the data in Table 1.

No. of Basic Blocks (n)	New Algo. (sec)	Stockmeyer's Algo. (sec)
100	0.150	0.101
200	0.339	0.363
300	0.546	0.786
400	0.760	1.374
500	0.979	2.118
600	1.212	3.022
700	1.448	4.082
800	1.680	5.292
900	1.926	6.680

Table 1: Simulation result for unbalanced floorplans.

References

[1] M. Ben-Or, "Lower bounds for algebraic computation trees," In *Proc. 15th Annual ACM Symposium on Theory of Computing*, pp. 80–86, 1983.

[2] C. H. Chen and I. G. Tollis, "Area optimization of spiral floorplans," *Journal of Circuits, Systems and Computers*, Vol. 3, No. 4, pp. 833–857, 1993.

[3] K. Chong and S. Sahni, "Optimal realizations of floorplans," *IEEE Trans. on CAD*, Vol. 12, No. 6, pp. 793–801, 1993.

[4] W.-M. Dai and E. S. Kuh, "Simultaneous floor planning and global routing for hierarchical building block layout," *IEEE Trans. on CAD*, Vol. 6, No. 5, pp. 828–837, 1987.

[5] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, New York, Wiley, 1990.

[6] R. H. J. M. Otten, "Automatic floorplan design," in *Proc. 19th DAC*, pp. 261-267, 1982.

[7] P. Pan and C. L. Liu, "Area minimization for floorplans," *IEEE Trans. on CAD*, Vol. 14, No. 1, pp. 123–132, 1995.

[8] P. Pan, W. Shi and C. L. Liu, "Area minimization for hierarchical floorplans," in *Proc. ICCAD,* pp. 436-440, 1994, to appear in *Algorithmica*.

[9] M. Sarrafzadeh, "Transforming an arbitrary floorplan into a sliceable one," in *Proc. ICCAD*, pp. 386–389, 1993.

[10] L. Stockmeyer, "Optimal orientations of cells in slicing floorplan designs," *Information and Control*, Vol. 57, pp. 91-101, 1983.

[11] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM Press, Philadelphia, PA, 1983.

[12] A. K. Tsakalidis, "AVL-trees for localized search," *Information and Control*, Vol. 67, pp. 173–194, 1985.

[13] T.-C. Wang and D. F. Wong, "Optimal floorplan area optimization," *IEEE Trans. on CAD*, Vol. 11, No. 8, pp. 992-1002, 1992.

[14] S. Wimer, I. Koren, and I. Cederbaum, "Optimal aspect ratios of building blocks in VLSI," *IEEE Trans. on CAD*, Vol. 8, No 2, pp. 139–145, 1989.

[15] G. Zimmermann, "A new area and shape function estimation technique for VLSI layout," in *Proc. 25th DAC*, pp. 60–65, 1988.