Symmetries in Rectangular Block-Packing

Hay-Wai Chan[1] and Igor L. Markov[2]

[1] hhchan@umich.edu
[2] imarkov@eecs.umich.edu
Department of Electrical Engineering and Computer Science, University of Michigan

Abstract. The block-packing problem has applications in chip design, factory layout, container shipment optimization, scheduling and other areas. It has been actively studied in the last 20 years, but symmetry in packings has been neglected. Our work deals with symmetry in several formulations of block-packing, including optimization and constraint-satisfaction variants, as well as slicing and non-slicing packings. We establish the presence of symmetries and show that symmetry-breaking improves the performance of our optimal block-packer BloBB, that is based on branch-and-bound and can solve small instances of the problem. Using this optimal solver we compare the efficiency of symmetry-breaking in different variants of block-packing.

1 Introduction

The rectangular block-packing problem has been extensively studied in the context of VLSI design. In short, the block-packing problem is to find a packing of a given set of rectangular blocks such that they do not overlap and the area of the bounding rectangle is minimized. A constraint satisfaction variant is to pack the blocks so that the dead space is smaller than a threshold value. An alternative constraint satisfaction version is to pack the blocks into a predefined box [1, 2, 4, 5, 7]. Since the block-packing problem is NP-hard [9], simulated annealing is typically used in applications. The use of symmetries holds a potential to simplify the block-packing problem by shrinking the solution space, but local optimization heuristics are not well-suited to use symmetry in search [11]. In fact, the block-packing problem turns out to be highly symmetrical as we show in Section 3. While the problem is highly symmetrical, practical instances often have instance-specific symmetries. In this paper, we discuss both instance-independent and instance-specific symmetries. We propose an optimal branch-and-bound block-packer that can pack up to 9 blocks with randomly-chosen dimensions in a reasonable amount of time, or even more in the presence of instance-specific symmetry.

General block-packing is often difficult to analyze. In this work we particularly consider *slicing* packings whose bounding rectangles can be recursively bisected by horizontal or vertical cuts into rooms, each of which contains one block. Slicing packings may not capture the best solutions, but many VLSI designers still rely on them as they lead to faster algorithms without significant quality losses (as well as for certain engineering reasons). Our work makes the case for slicing packings even stronger by presenting simple yet effective symmetry-breaking strategies. They significantly speed up the optimal branch-and-bound slicing packer so that it can solve up to 12 blocks with random dimensions in a reasonable amount of time.

The block-packing problem is usually formulated and studied as an optimization problem, as it appears in VLSI design. We present its alternative formulation as a constraint satisfaction problem, where our symmetry-breaking techniques also apply.

The rest of the paper is as follows. Preliminaries are given in Section 2 and motivation for our work in Section 3. Sections 4 and 5 describe our optimal non-slicing and optimal slicing

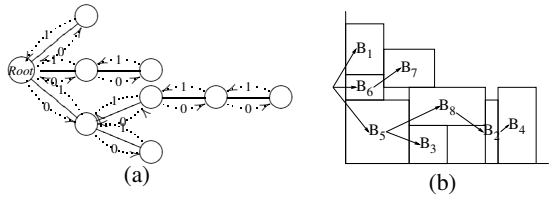


Fig. 1. (a) The tree represented by $T = 0010001111001101$; (b) the packing (T, π, θ) where $\pi = \{B_5, B_3, B_8, B_2, B_4, B_6, B_7, B_1\}$.

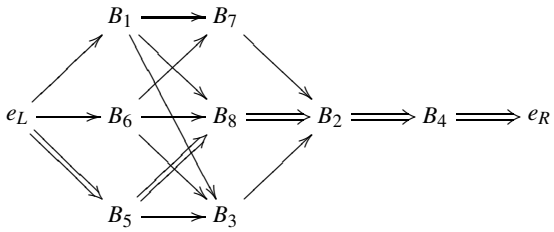


Fig. 2. Horizontal constraint graph of the packing in Fig.1b. The horizontal critical path is indicated by the double arrows. Transitive edges are omitted for clarity.

block-packers respectively. Section 6 discusses the enumeration of symmetric packings. We discuss empirical results in Section 7 and conclude in Section 8.

2 Preliminaries

The Rectangle Packing Problem. Let $M = \{B_1, \ldots, B_m\}$ be a set of rigid rectangular blocks. A *packing* of M defines, for every block B_i, its orientation θ_i and planar location (x_i, y_i). No two blocks may overlap. One seeks to minimize the area of the bounding box of the floorplan. This formulation outlines an optimization problem. A constraint satisfaction version of the problem is to pack the set of blocks such that the dead space is smaller than a threshold value.

Conventions and Background. In the rest of the paper, the term *non-slicing* means "not necessarily slicing". All sets are ordered. A permutation of order n is just an ordered n-element set, typically of blocks $\{B_1, \ldots, B_n\}$. This defines a precedence relation $\prec$ on blocks, which are often referred to by indices, e.g., 2 may denote block B_2. To know the width w_B and height h_B of block B, one needs to know its orientation. The location of B is the location of its bottom-left corner. As is common for constraint graphs, we consider the left edge of the core region e_L and its bottom edge e_B as two fixed hypothetical blocks ("poles"). e_L has width 0 and height ∞, while those of e_B are ∞ and 0 respectively.

Constraint Graphs and Critical Paths. Given a packing U of n blocks with specified orientations, its *horizontal constraint graph* (*HCG*) G_H is a graph (V, E, f) such that the set of vertices $V = \{B_1, \ldots, B_n, e_L, e_R\}$ where e_L (e_R) is the left (right) edge of the bounding rectangle of U, $E = \{(v_1, v_2) \,|\, v_1 \text{ is in the left of } v_2\}$ and the weight of B_i, $f(B_i) = w_{B_i}$ and $f(e_L) = f(e_R) = 0$ (Fig.2). It is clear G_H is a directed acyclic graph with source e_L and sink e_R. Similarly for *the vertical constraint graph* (*VCG*) of U with top edge e_T and bottom edge e_B and the weight of B_i is its height. There may be multiple longest paths in the HCG or VCG of U, we declare the lexicographically smallest one to be *the* longest path. We refer to the longest

path in HCG (VCG) of U as the *horizonal critical path* (*vertical critical path*), abbreviated as $hcp(U)$ ($vcp(U)$). If $hcp(U) = (k_1, \ldots, k_r)$, we define *the horizontal critical path set* of U, $hcps(U)$, as $\{k_1, \ldots, k_r\}$. Similarly for *the vertical critical path set* of U, $vcps(U)$.

The O-tree Representation. We choose to use the O-Tree representation for non-slicing floorplans since its solution space is the smallest among all proposed. Moreover, as we will see in Section 4.2, a partial O-Tree provides useful information for effective pruning. A rooted ordered tree with $n+1$ nodes can be represented by a bit-vector of length $2n$, which records a DFS traversal of the tree. 0 and 1 record downward and upward traversals respectively (Fig.1a). An O-Tree for n blocks is a triplet (T, π, θ) where T is a bit-vector of length $2n$ specifying the tree structure, π is a permutation of order n listing the blocks as they are visited in DFS, θ is a bit-vector of length n with block orientations (0 for "not rotated" and 1 for "rotated by $\pi/2$").

(T, π, θ) represents a packing by sequencing its blocks according to π. The x-coordinate x_B of a newly-added block B is 0 if its parent P is the root of T (e_L), or else $x_P + w_P$, the sum of the width of P (implied by θ) and its x-coordinate. The y-coordinate y_B is the smallest non-negative value that prevents overlaps between B and blocks appearing before B in π (Fig.1b). e_B can be a child of any leaf.

A packing is *L-compact* (*B-compact*) iff no block can be moved left (down) while other blocks are fixed. A packing is *LB-compact* iff it is both L-compact and B-compact (equivalently defined as *admissible* in [6] and *maximally compact* in [8]). The packing in Fig.1b is LB-compact. Every LB-compact packing can be represented by an O-Tree, and all packings specified by an O-Tree are B-compact [6]. The y-coordinate for each block can be found in amortized $O(1)$ time, facilitating the realization of an O-Tree with n blocks in $O(n)$ time [6].

Normalized Polish Expressions (NPEs). NPE is a non-redundant representation for slicing floorplans [12] and as we will see in Section 5, its solution space is very easy to be enumerated. cc A *slicing floorplan* is a rectangle area recursively sliced by horizontal and/or vertical cuts into rectangular rooms [8]. A packing is *slicing* if its bounding rectangle is a slicing floorplan and each rectangular room contains exactly one block. Slicing packings can be represented by slicing trees. Each leaf node of a slicing tree represents a block and each internal node represents a horizontal or vertical cut (Fig.3). We can also consider each internal node to be a *supermodule*, consisting of the two blocks or supermodules represented by its children and merged in the way specified by itself. A sequence $\alpha_1 \ldots \alpha_{2n-1}$ over $\{1, \ldots, n, *, +\}$ is said to be a *Polish expression* if there is a slicing tree T such that $\alpha_1 \ldots \alpha_{2n-1}$ is the sequence of nodes visited in a post-order traversal of T. It is *normalized* if it does not contain consecutive $+$'s or $*$'s. For example, the expression in Fig.3c is normalized, but that in Fig.3b is not. The above correspondence defines a bijection between the set of slicing trees with n leaves and the set of Polish expressions of length $2n-1$. The set of normalized Polish expressions of length $2n-1$ is in a 1-1 correspondence with the set of slicing floorplans with n blocks [12].

Given a slicing tree T and the orientations of the blocks, *the slicing packing of* T is a packing specified by T such that no vertical (horizontal) cuts can be moved to the left (down), and each block is placed at the bottom-left corner of the room (Fig.3). Operators $+$ and $*$ act on the set of blocks $\{1, \ldots, n\}$ and supermodules such that $A + B$ ($A * B$) is the supermodule obtained by placing B on top of (to the right of) A. Polish expressions use the postfix notation for such operators. To evaluate a floorplan, we can simply compute the supermodule that contains all blocks by recursively merging blocks and supermodules. This procedure can be implemented in $O(n)$ time and will be explained in Section 5.1.

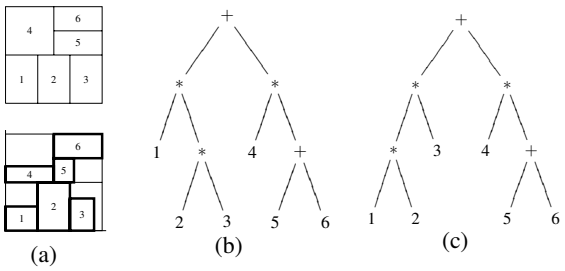


Fig. 3. (a) A slicing floorplan and a slicing packing; (b) a slicing tree representing (a), its Polish expression is $123**456++$; (c) an equivalent slicing tree whose Polish expression is $12*3*456++$.

Tree T: 1 4 7 ····· 2 bits each time
Permutation π: 2 5 8 ····· 1 block each time
Orientation θ: 3 6 9 ····· 1 bit each time

Fig. 4. Branching Schedule

3 Motivation

Consider packing U of n blocks $\{B_1,\ldots,B_n\}$ with specified orientations. If U is slicing or specified by an O-Tree, then the width of U equals to the length of $hcp(U)$ and hence the sum of widths of blocks in $hcps(U)$. For another slicing or LB-compact packing U' such that $hcps(U') = hcps(U)$, the widths of U' and U are equal. Since $\emptyset \neq hcps(U) \subseteq \{B_1,\ldots,B_n\}$, there are at most (2^n-1) values of its width and similarly its height. Therefore, there are at most 4^n possible width-height combinations for U. Since there are 2^n possible combinations of orientations of $\{B_1,\ldots,B_n\}$, there are at most 8^n possible width-height combinations. Note that the actual number of width-height combinations is much smaller than 8^n since $|hcps(U) \cap vcps(U)| = 1$ if U is slicing or specified by an O-Tree. However, even 8^n is very small compared with the size of solution space of O-Tree which is $O\left(n!\frac{2^{3n-2}}{n^{1.5}}\right)$ or that of NPEs [13].

Two packings are *symmetric* if they have the same shorter edges and longer edges. Obviously, this relation is an equivalent relation. For example in Fig.6, packings P, P' and P'' are symmetric but packings P and P''' are not. To minimize area, we only have to consider one packing from each equavalent class. Therefore, effective symmetry-breaking may, in principle, reduce search runtime, but that requires careful tracking of symmetry that would be lightweight enough not to spoil the speed-ups provided by solution-space reduction. Handling all symmetries is not necessary, and one is free to choose the types of symmetry that can be handled efficiently. However, the more types of symmetry yield to such techniques, the greater speed-ups may be achieved. As we will see in Section 6, enumerating symmetric packings is useful for optimizing objectives other than area.

4 Optimal Non-slicing Block-Packing

The floorplan representation used by a branch-and-bound floorplanner may critically affect its performance. We choose the O-Tree representation because no known representation

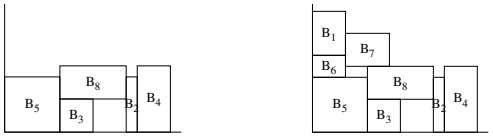


Fig. 5. (a) A partial packing (t,σ,δ) with $t = 0010001111$ and $\sigma = \{B_5, B_3, B_8, B_2, B_4\}$, (b) a compatible complete packing.

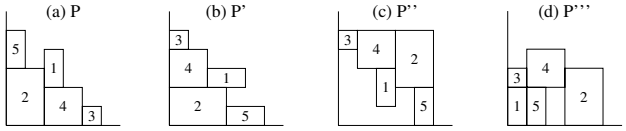


Fig. 6. The packing P satisfies (4.2) but not (4.1). When we apply an α-transformation to get P', P' does not satisfy (4.2) anymore. Thus we apply a β-transformation to get P''' by flipping P' to P'' and then compacting to P'''.

achieves a smaller amount of redundancy. A partial O-Tree defines a partial packing that can be extended to complete packings, and this property facilitates effective pruning.

4.1 Branching

BRANCHING SCHEDULE. We adopt a branching schedule in Fig.4 such that at each layer of the search tree, we define 2 bits of T, 1 block of π, or 1 bit of θ. Our basic framework is a depth-first search.

A bit-vector identifies a rooted ordered tree iff it has equal numbers of 0's and 1's and every prefix has at least as many 0's as 1's. Hence, a partial bit-vector t with i 0's and j 1's can be extended to one representing a rooted ordered tree with n nodes iff (1) $i \geq j$ and (2) $i \leq n$. These *feasibility* conditions can be easily checked in $O(1)$ time upon every incremental change to the bit-vector. Infeasible bit-vectors are pruned, and we may get a new feasible bit-vector t at every search node of depth $4i$.

INFORMATION IN A PARTIAL SOLUTION. Suppose (T,π,θ) is extended from (t,σ,δ). Since t has at least i 0's, the positions of all blocks in σ in T are set. Furthermore, since δ is as long as σ, the orientations of all blocks in σ are determined. The position of a block in (T,π,θ) depends only on itself and its preceding blocks in π [6]. We can then determine the locations of all blocks in σ before we explore deeper and (t,σ,δ) determines *a partial packing*. Addition and deletion of a block take amortized $O(1)$ time [6] (Fig.5). We say (T,π,θ) to be *extended* from (t,σ,δ) if t, σ, and δ are prefixes of T, π, and θ respectively. It is *an extended packing* of (t,σ,δ).

4.2 Symmetry and Dominance-breaking

In subsequent discussions, we consider a partial packing $U = (t,\sigma,\delta)$ of i blocks and an extended packing (T,π,θ) of n blocks. Let m_k be the length of the shorter edge (min-edge) of block k for $k = 1 \ldots n$. Let $\{C_0, \ldots C_{m+1}\}$ be the contour of U. We do not distinguish between T and the *tree* presented by T, and similarly for t. The estimations of dead space lower bound

are quite involved and omitted for brevity. Their descriptions and justifications are available upon request.

DOMINANCE-BREAKING. The bounding rectangle of a packing can be in one of eight orientations. It suffices to analyze only one of those orientations. To facilitate pruning, observe that an LB-compact packing always contains unique lower-left, lower-right and upper-left blocks, and at least one upper-right block. We declare the rightmost upper-right block to be *the upper-right block*. In Fig.1b, B_4 is the upper-right block. To avoid redundant packings, we impose dominance-breaking constraints:

(4.1) the lower-left block $B_{lower-left}$ has orientation 0,
(4.2) $B_{lower-left} \preceq R$ for every corner block R.

We enforce constraint (4.1) because any packing can be flipped across the diagonal, preserving the lower-left block. Enforcing that constraint by applying a diagonal flip is referred to as α-*transformation* (Fig.6a-b).

Similarly, we can transform $P = (T, \pi, \theta)$ to an LB-compact packing P' satisfying constraint (4.2) by a series of flips and/or rotations, followed by rounds of L- and B-compaction. The height and width of P' will not exceed those of P. This transformation of packings is referred to as β-*transformation* and illustrated in Fig.6b-d. Note that compaction is necessary to keep the resultant packing inside the solution space of O-Tree.

To enforce constraints (4.1-2) simultaneously, we may have to apply an α-transformation followed by a β-transformation, and then repeat. Constraints (4.1-2) will be satisfied in at most n such steps since the index of the lower-left block decreases after a β-transformation and stays the same after an α-transformation.

Packing U is said to *dominate* packing U' if the sides of U are not longer than those of U'. If we consider symmetric packings to be equal, then the relation of dominance defines a partial ordering among the packings. Obviously, the resultant packing after applying rounds of α and β-transformations dominates the original one, which we can safely prune.

Let $M_\sigma = \max_{k \notin \sigma}\{k\}$ and I_{lr} be the index of the current lower-right block. The index of the lower-right block is at most $I = \max(I_{lr}, M_\sigma)$. Since lower-left block in the partial packing must remain the lower-left block in any of its extended packings, we require $B_{bottom-left} \preceq B_I$. Similarly for upper-left and upper-right blocks. For every packing with more than one block, the upper-left block and lower-left block must be distinct. Thus, if $B_{lower-left}$ has index larger than both the current upper-left and lower-right blocks, we can require $B_{lower-left} \prec S$ where the index of S is the second largest among all unused blocks. In special cases when the contour lines form a single horizontal line, we can require $B_{bottom-left} \preceq R$ for every corner block R of the partial packing. Dominance-breaking by constraint (4.1) is instance independant while that by constraint (4.2) is instance specific.

BLOCKS WITH SAME HEIGHT OR WIDTH. If two adjacent blocks B and B' have the same height and y-coordinate, the cluster formed by B and B' can be flipped. We break this symmetry by requiring $B \prec B'$ if B is to the left of B'. A similar constraint applies to adjacent blocks with same width and x-coordinate, e.g., B_1 and B_6 in Fig.1b. If two blocks B_i and B_j in π have the same width and height ($i < j$), they are interchangeable. We require B_i to appear first in σ.

These constraints are compatible with constraints (4.1-2) since the index of lower-left block does not increase while those of other corner blocks do not decrease after flips introduced above. It is clear that is type of symmetry is instance-specific.

expression p: 1 3 5 7 8 10
orientation δ: 2 4 6 9

Fig. 7. Branching schedule towards $(124*5+,0111)$.

(a)	(b)	(c)
expression: 1 2	1 2 4	1 2 4 *
bundle: B_1 B_2	B_1 B_2 B_4	B_1 M_{24*}
storage:		B_4 B_2

Fig. 8. (a) The original configuration; (b) adding 4 to (a); (c) adding $*$ to (b); removing $*$ from (c) yields (b); removing 4 from (b) yields (a).

5 Optimal Slicing Block-Packing

We use the Normalized Polish Expression (NPE) representation since it is a non-redundant, bottom-up description of slicing packings, with incremental addition/deletion of blocks and easy enumeration.

5.1 Branching

BRANCHING SCHEDULE. A slicing packing of n blocks can be specified by (P,θ) where P is a Polish expression of length $2n-1$ and θ is a bit-vector of length n, storing the orientations of the blocks as described in Section 2. We maintain a growing Polish expression p and bit-vector δ.

We explore symbols of p one by one. If a given symbol is an operand, we explore a bit of δ, otherwise another symbol of p is explored (Fig.7). We use the following characterization of Polish expression. A sequence $z_1 \ldots z_{2n-1}$ over $\{1,\ldots,n,+,*\}$ is a Polish expression iff for every $i = 1,\ldots,n$, i appears exactly once in the sequence and every prefix has more operands than operators [12]. As a corollary, a sequence p over $\{1,\ldots,n,+,*\}$ of length $m \leq 2n-1$ can be extended to a normalized Polish expression iff (1) for every $i = 1,\ldots,n$, i appears at most once in p, (2) p has more operands than operators and (3) there are no consecutive $+$'s and $*$'s in p. Such sequences are called *partial Polish expressions*, and can be tested for in $O(1)$ time per incremental change.

INFORMATION IN A PARTIAL SOLUTION. We maintain a series of blocks and supermodules using two stacks: *the bundle* and *the storage*.

When we push an operand and its orientation to p and δ respectively, we push the respective block (with width and height specified) into the bundle stack. When we push an operator α to p, we are guaranteed to have at least two blocks or supermodules in the bundle. We pop the two top-most blocks in the bundle, A and B, and push them in this order into the storage. We compute the supermodule formed by merging A and B in the way specified by α and push it into the bundle. When we pop an operand b and its orientation from p and θ respectively, we pop the top element of the bundle, which is necessarily b. When we pop an operator α from p, we pop the top element of the bundle, and push the two top-most blocks or supermodules from the storage to the bundle (Fig.8). During incremental changes to p and δ, stack updates take $O(1)$ time. When we reach a leaf of the search tree, the supermodule in the bundle is the bounding rectangle specified by a complete solution (P,θ).

5.2 Symmetry-breaking

For two supermodules (or blocks) M and N, we define $M \prec N$ if $B_M \prec B_N$ where B_M and B_N are the bottom-left blocks of M and N respectively. For two supermodules (or blocks) A and B, we define $A+B$ as the supermodule formed by placing B on top of A, and $A*B$ as that formed by placing B in the right of A. When we consider two partial Polish expressions, we implicitly assume that they are associated with the same bit-vector δ and hence represent two packings. The estimations of area lower bound are omitted for brevity.

COMMUTATIVITY. $A+M$ is equivalent to $M+A$, and $A*M$ to $M*A$. To break this symmetry when merging supermodules A and M, one can require $A \prec M$. We propose a better pruning mechanism below.

Suppose we are pushing the block B to the bundle, which is not empty, with the top element A. Then B must be the bottom-left block of the next supermodule M to merge with A. Hence we require $A \prec B$, implying an ascending order of blocks and supermodules in the bundle. This type of symmetry is instance-independent since we impose no restriction on the relative positions of the blocks.

ABUTMENT. Consider blocks R_1, R_2 and R_3, where $R_1 \prec R_2 \prec R_3$. If they abut horizontally or vertically, their order does not matter. For example, $(R_1+R_3)+R_2$ is equivalent to $(R_1+R_2)+R_3$. However both arrangements pass the commutativity constraint.

For chained operators of the same kind, e.g., $(R_1+R_2)+R_3$ or $(R_1*R_2)*R_3$, we require both $R_1 \prec R_3$ and $R_2 \prec R_3$. By the commutativity constraint $R_1 \prec R_2$. Therefore we only have to check if $R_2 \prec R_3$. Since an abutment of three or more blocks must be of the form $E_1E_2+E_3+\ldots+E_i+$, the abutment constraint breaks *all* symmetries of this kind. Similar to the commutivity constraint, this type of symmetry is instance-independent.

GLOBAL BOTTOM-LEFT BLOCK AND ITS ORIENTATION. We require B_1 to be the bottom-left block of all packings. This constraint is redundant because the commutativity constraint does not allow pushing B_1 to a non-empty bundle. However we can now prune hopeless partial Polish expressions much sooner. Similar to the non-slicing case, we require the orientation of B_1 to be 0.

IDENTICAL BLOCKS. If blocks A and B have the same dimensions, then they are interchangeable. Since the above constraints do not break all symmetries due to identical blocks, we require in that case that A appear before B in p if $A \prec B$. Similar to the non-slicing case, this type of symmetry is instance-specific.

6 Enumeration of Symmetric Packings

We skip through symmetric packings to speed up our branch-and-bound search, but there are cases where symmetric packings are helpful. We can first optimize area by finding an optimal or a sub-optimal packing and then minimize the secondary objective among all symmetric solutions. For example in VLSI floorplanning, we can minimize wirelength among solutions of the same area. High-quality solutions can be found if this solution space captures a wide range of packings.

Permutations of identical blocks are the simplest kinds of symmetry. We can always interchange identical blocks regardless their positions and orientations, and hence easily describe the rearrangements as permutations of blocks. Suppose we partition $S=\{B_1,\ldots,B_n\}$ into $S_1,\ldots,S_r$ such that $\cup_{i=1}^{r} S_i = S$, $S_i \cap S_j = \emptyset$ for all $i \neq j$ and all blocks in S_i are identical. The group $G_S = Sym_{S_1} \times \ldots \times Sym_{S_r}$ contains *all* symmetries by identical blocks, where Sym_T

is the symmetric group of T. When all blocks are distinct, $|S_i| = 1$ for all $i = 1, \ldots, n$ and $G_S \cong \{1\}$. On the other hand, when all blocks are identical, $G_S = Sym_S \cong Sym_n$. The group $Z_2 = \{0, 1\}$ describes symmetry by flipping the packing preserving the bottom-left block, where 1 represents a flip. Hence, for a set of blocks S, its *identical block group* $G_S \times Z_2$ describes all symmetries by identical blocks and flipping, regardless how the blocks are packed.

6.1 Enumerating Optimal Non-slicing Packings

α-transformations in Section 4.2 are invertible and are represented by the group Z_2 as mentioned above. However, β-transformations are not invertible since they involve compaction. Multiple packings may be mapped to the same packing by rounds of α and β-transformations. Therefore it is non-trivial to recover all packings that give the optimal one. Moreover, some of these packings may have sub-optimal areas. We only consider the eight orientations of the packing, possibly with compaction.

We only have to consider four orientations of the packing that preserves the orientations of the blocks, since applying α-transformations can generate the rest. We can construct O-Tree that compact blocks to other corners of the bounding rectangle in $O(n)$ time [10]. Further compacting blocks takes at most $O(n^2)$ time [6]. Since we are working with an optimal packing, compaction does not shorten the sides and the resultant packing is symmetric to the original one. Therefore, we can get the eight orientations for the packing in $O(n^2)$ time. In special cases where i blocks of the same height abut horizontally and share the same y-coordinate, we can permute them. Similarly when they abut vertically and have the same width.

As we see in Section 7, this strategy recovers many symmetric packings. Note that some of the above transformations among symmetric packings are not necessarily invertible, and hence they do not form a group in general.

6.2 Enumerating Symmetric Slicing Packings

Unlike in the non-slicing case, symmetric floorplans can be recovered very easily. Given a slicing tree T, we can analyze its symmetric floorplans by its pre-order traversal. A slicing tree is *normalized* if its equivalent Polish expression is normalized. It is clear that a slicing tree is normalized iff no internal nodes share the same sign as its right child. A sequence $X = x_1 \ldots x_{2n-1}$ over $\{1, \ldots, n, +*\}$ is said to be a *prefix expression* if there exists a (unique) slicing tree T whose pre-order traversal gives X. X is said to be *normalized* if T is normalized. For example the prefix expression of the slicing tree in Fig.3b is $+*1*23*4+56$ and of that in Fig.3c is $+**123*4+56$. The latter is normalized but the former is not. The set of normalized prefix expressions is in 1-1 correspondence with the set of normalized slicing trees, the set of NPEs and the set of slicing packings.

Consider a slicing packing, its NPE P and its normalized prefix expression X. A horizontal abutment of i blocks, $i \geq 2$, corresponds to the partial Polish expression $E_1E_2*\ldots E_i*$ in P where E_j is the partial Polish expression representing the abutting block or supermodule. Therefore, it corresponds to a sequence of $(i-1)$ consecutive $*$'s ($*^{i-1}$) and partial prefix expression $*^{i-1}E_1E_2\ldots E_i$ in X. Similarly for $+$'s. For example in Fig.3c, the abutment of blocks 1, 2 and 3 corresponds to partial Polish expression $12*3*$ and partial prefix expression $**123$. Conversely, the sequence $*^{i-1}$ in a normalized prefix expression correspond to a horizontal abutment of i blocks. Similarly for $+$'s.

For a normalized prefix expression X for n blocks, we define its *characteristic sequence* to be $(z_1^1 \ldots z_1^{k_1}, z_2^1 \ldots z_2^{k_2}, \ldots, z_r^1 \ldots z_r^{k_r})$, $r \geq 1$ where,

1. $z_j^1 = z_j^2 = \ldots = z_j^{k_j} \in \{+, *\}$ for $j = 1, \ldots, r$. $z_j^1 \ldots z_j^{k_j}$ is called *a chain*.
2. z_j^i appears right before z_j^{i+1} in X for $i = 1, \ldots, k_j - 1$, $j = 1, \ldots, r$.
3. $z_j^{k_j}$ appears before z_{j+1}^1 in X. If $z_j^{k_j}$ appears right before z_{j-1}^1, then $z_j^{k_j} \neq z_{j+1}^1$ for $j = 1, \ldots, r-1$.
4. $k_1 + \ldots + k_r = n - 1$.

$(k_1, k_2, \ldots, k_r)$ is said to be a *signature* of X. It is clear that every normalized prefix expression has a unique characteristic sequence and signature. For example in Fig.3c, the characteristic sequence is $(+, **, *, +)$ and signature is $(1, 2, 1, 1)$. The characteristic sequence and signature indicate which partial prefix expressions in X or equivalently which subtrees in the slicing tree of X are interchangeable. The i partial prefix expressions $E_1, \ldots, E_i$ after the chain $*^{i-1}$ or $+^{i-1}$, $i \geq 2$ can be rearranged in any arbitrary order, preserving the dimensions of the supermodule that they form. For example in Fig.3c, the partial prefix expressions 1, 2 and 3 after the chain $**$ can be exchanged in any order. Similarly we can swap the subtrees 4 and $+56$ after the chain $*$.

Due to the hierarchical structure of slicing floorplans, rearranging subtrees about each chain in the slicing tree is independent of each other. For example in Fig.3c, rearranging subtrees 1, 2 and 3 about the chain $**$ is independent of swapping 4 and $56+$ about the chain $*$. Therefore, given a normalized prefix expression X with characteristic sequence $(z_1^1 \ldots z_1^{k_1}, \ldots, z_r^1 \ldots z_r^{k_r})$ and signature $(k_1, \ldots, k_r)$, there are $(k_j + 1)!$ ways to rearrange the $(k_j + 1)$ partial prefix expressions after the chain $+^{k_j+1}$ or $*^{k_j+1}$, and equivalently rearrange the corresponding subtrees in its slicing tree. We recover symmetry by commutativity constraint when $k_j = 1$ and abutment constraint when $k_j \geq 2$.

Given a slicing floorplan of n blocks, its NPE P, its normalized prefix expression X with characteristic sequence $\left(z_1^1 \ldots z_1^{k_1}, \ldots, z_r^1 \ldots z_r^{k_r}\right)$, and normalized slicing tree T, we define its *structural group* as

$$G_P = \left\{(\sigma_1, \ldots, \sigma_r) | \sigma_j \text{ permutes the partial prefix expressions after the chain } z_j^1 \ldots z_j^{k_j} \text{ in } X\right\}$$

We can also think of σ_j as the operation which permutes the $k_j + 1$ subtrees under the chain of k_j nodes in T. For example in Fig.3c, $\left(\begin{bmatrix} 12*3* \rightarrow 456+* \\ 456+* \rightarrow 12*3* \end{bmatrix}, \begin{bmatrix} 1 \rightarrow 3 \\ 2 \rightarrow 2 \\ 3 \rightarrow 1 \end{bmatrix}, \begin{bmatrix} 4 \rightarrow 56+ \\ 56+ \rightarrow 4 \end{bmatrix}, \begin{bmatrix} 5 \rightarrow 5 \\ 6 \rightarrow 6 \end{bmatrix}\right) \in G_{12*3*456+*+}$. It is not hard to see that $G_P \cong Sym_{k_1+1} \times \ldots \times Sym_{k_r+1}$. By counting, $|G_P| = \prod_{j=1}^{r} (k_j + 1)!$. If the signature of X is $(1, 1, \ldots, 1)$, then $|G_P| = 2^{n-1}$. On the other hand, if the signature of X is $(n-1)$, then $|G_P| = n!$. They are the lower and upper bounds of the size of G_P respectively. G_P can be computed in $O(n)$ time since only a pre-order traversal is needed. *All* symmetric floorplans can be recovered since only the commutativity and abutment constraints are imposed. Note that this group acts on the set of partial Polish expressions of P *faithfully*, implying that the size of G_P equals to the number of symmetric floorplans generated.

7 Experimental Results

Our algorithms are implemented in C++ and will be open-sourced under the name of BloBB (Block-packing with Branch-and-Bound). All programs are compiled with `g++ 3.2.2 -O3` and evaluated on a Linux Anthlon $1.2GHz$ workstation. Table 1 shows its runtimes on

randomly-generated test cases and test cases with only 2 or 3 types of blocks. The randomly-generated test cases are more difficult for our block-packer since there is no symmetry among the blocks. On the other hand, the 2 and 3-block-type test cases are easier since symmetry-breaking (SB) by identical blocks is very effective. Observe that the dead space (%) in optimal packings *decreases* in larger floorplans, and the dead space in test cases with 2 and 3 block-types is significantly smaller than the randomly-generated test cases.

BloBB's performance on the three smallest MCNC benchmarks (available from [14]) is shown in Table 4. We compare its runtimes with full symmetry-breaking and limited symmetry-breaking. We always impose symmetry-breaking constraints by identical blocks and orientation of the bottom-left block since BloBB does not terminate in a reasonable amount of time without them. (We also apply symmetry-breaking by global bottom-left block for optimal slicing block-packer.) Our non-slicing block-packer runs 2.0 to 3.0 times faster with symmetry-breaking constraints from Section 4.2. Note that those constraints are more effective when there are fewer identical blocks. For example, they are more effective on *xerox* than *apte* and *hp*. It is because symmetry by identical blocks sometimes overlaps with symmetry by other constraints such as symmetry by blocks with the same edge. We can see that the speed-up ratio is high when the size of identical block group is small. Similarly for the slicing case, where the effect of symmetry-breaking is even more significant.

We modified BloBB slightly to handle the constraint satisfaction version of the block-packing problem. Table 4 shows that the symmetry-breaking techniques for area minimization version also improve the performance of the modified BloBB. The improvement is less significant in the non-slicing case since our symmetry-breaking constraints mainly prune away many lexicographically large solutions and few lexicographically small solutions. Since we are likely to find a lexicographically small solution in the constraint satisfaction version of the problem, the effect of symmetry-breaking is expected to be not as pronounced as in area minimization. However, the improvement is still significant. In contrast, symmetry-breaking improves the performance of slicing constraint satisfaction problem much more significantly. It is because the solutions pruned by symmetry lie relatively evenly throughout the whole solution space where the solution starts with block 1.

Another way to evaluate the symmetry-breaking strategies is by enumeration. Only one optimal packing of a given width-height combination should be considered in all cases if the set of symmetry-breaking constraints is complete. We modify our branch-and-bound algorithm to enumerate all optimal slicing and non-slicing packings that it considers as non-symmetric. From Table 3, our symmetry-breaking strategies for slicing packings are very effective in practice. Only one packing is generated in each of the test cases. Fig.9 shows the optimal packings produced by BloBB. In optimal non-slicing packings of *hp* and *xerox*, our symmetry-breaking can be further improved by identifying slicing subfloorplans inside non-slicing packings. For example in Fig.9a, slicing symmetry-breaking strategies can be applied to the cluster consisting of blocks 2, 9 and 11.

8 Conclusions and Ongoing Work

We formulate the rectangular block-packing problem as an optimization problem and a constraint satisfaction problem. We implement optimal block-packers for slicing and non-slicing packings and show how symmetry-breaking improves their performance. Table 5 summarizes our dominance and symmetry-breaking techniques. Given a slicing packing, we can compute the group that restore all symmetric packings at $O(n)$ time.

Table 1. BloBB runtimes.

	optimal non-slicing						optimal slicing					
	random		3 block-types		2 block-types		random		3 block-types		2 block-types	
# of blocks	dead space % / runtime (s)		dead space % / runtime (s)		dead space % / runtime (s)		dead space % / runtime (s)		dead space % / runtime (s)		dead space % / runtime (s)	
6	4.12	0.24	2.72	0.043	1.88	0.014	5.51	0.015	3.63	0.009	2.48	0.002
7	3.52	2.25	2.16	0.19	1.20	0.030	4.85	0.057	2.55	0.014	1.32	0.009
8	3.07	38.4	3.02	1.35	1.10	0.20	4.49	0.29	3.30	0.068	1.30	0.026
9	2.48	664	1.89	8.06	1.68	1.19	3.81	1.54	2.05	0.16	1.91	0.15
10	—	—	1.96	46.9	1.74	4.20	3.90	28.0	2.20	0.88	1.99	0.45
11	—	—	—	—	0.91	19.3	3.52	96.2	1.68	6.49	1.08	1.09
12	—	—	—	—	0.96	83.7	3.16	545	2.22	12.9	1.08	2.85
13	—	—	—	—	—	—	—	—	2.13	30.9	1.52	17.9
14	—	—	—	—	—	—	—	—	1.94	131	2.39	46.4
15	—	—	—	—	—	—	—	—	1.87	617	0.94	63.0
16	—	—	—	—	—	—	—	—	—	—	1.29	309

Average performance of BloBB on 10 randomly-generated test cases. The dimensions are distributed uniformly in the range 1..200. All blocks in random test cases are distinct and the number of blocks in k-block-type test cases are as close to each other as possible.

Table 2. Information of *apte*, *xerox* and *hp*

test case	# of blocks	block area (mm^2)	block size distribution	size of identical block group
apte	9	46.562	4, 4, 1	1152
xerox	10	19.350	1, 1, 1, 1, 1, 1, 1, 1, 1, 1	2
hp	11	8.831	1, 1, 3, 2, 2, 2	96

Table 3. Number of optimal packings and their areas.

test case	optimal non-slicing				optimal slicing		
	area	no SB	limited SB	full SB	area	no SB	full SB
apte	46.925	725760	630	420	46.925	725760	1
xerox	19.796	104	52	5	20.017	1536	1
hp	8.947	5376	56	4	9.032	4608	1

Limited SB includes symmetry-breaking constraints mentioned in Table 4.

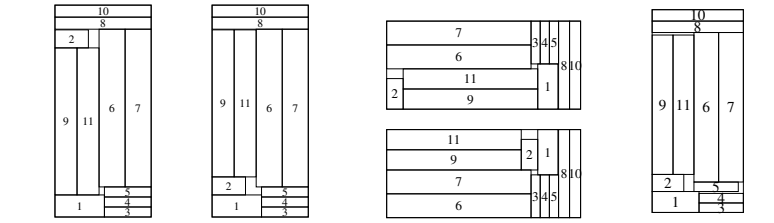
While symmetry-breaking for slicing packings is very close to being complete in practice, Table 3 shows that our symmetry-breaking for non-slicing packings is rather naive. As noted in Section 7, we believe that identifying slicing sub-packings inside non-slicing packings can eliminate many redundancies. Given a partial solution in the course of our branch-and-bound algorithm, we statically detect if all of its extended packings are dominated by some other packings. It is possible that we can improve the performance by adding a dynamical detection, even though it is unlikely to be computationally cheap. We also work towards an algebraic structure that describes relationships among a set of symmetric (optimal) non-slicing packings, compatible with our structural group for slicing packings.

Large instances of the block-packing problem can be solved hierarchically. One can first pack blocks into clusters and then pack clusters into higher-level clusters. Therefore, the performance of symmetry-breaking for small instances is relevant in practice. Recently

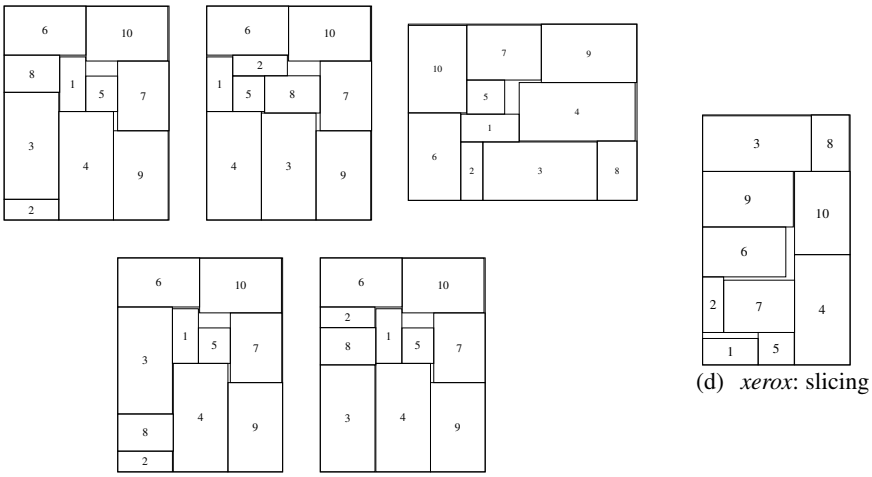
Table 4. BloBB performance results on *apte*, *xerox*, and *hp*.

	area minimization						constraint satisfaction					
	optimal non-slicing			optimal slicing			optimal non-slicing			optimal slicing		
test case	limited SB (s)	full SB (s)	speed up ratio	limited SB (s)	full SB (s)	speed up ratio	limited SB (s)	full SB (s)	speed up ratio	limited SB (s)	full SB (s)	speed up ratio
apte	6.03	2.35	2.57 : 1	2.19	0.210	10.4 : 1	0.000	0.000	—	0.67	0.080	8.38 : 1
xerox	33188	9812	3.38 : 1	1724	11.92	145 : 1	32.7	26.3	1.24 : 1	364	5.03	72.4 : 1
hp	2200	887	2.48 : 1	40.4	0.68	59.4 : 1	20.5	9.53	2.15 : 1	3.04	0.18	16.9 : 1

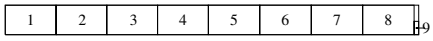
In the non-slicing case, limited SB includes symmetry-breaking by identical blocks and orientation of the bottom-left block and full SB includes dominance-breaking and symmetry-breaking by blocks with same height or width. In the slicing case, limited SB includes symmetry-breaking by identical blocks, orientation of bottom-left block and global bottom-left block and full SB includes symmetry-breaking by the commutivity constraints and the abutment constraints. In the constraint satisfaction test cases, BloBB finds the first solution with no more than 4% dead space.



(a) *hp*: non-slicing (b) *hp*: slicing



(c) *xerox*: non-slicing (d) *xerox*: slicing



(e) *apte*: slicing and non-slicing

Fig. 9. Optimal packings produced by BloBB.

Table 5. Summary of dominance and symmetry-breaking techniques.

non-slicing packings	α-transformation	instance-independent
	β-transformation	instance-specific
	blocks with same height or width	instance-specific
	identical blocks	instance-specific
slicing packings	commutivity constraint	instance-independent
	abutment constraint	instance-independent
	global bottom-left block and its orientation	instance-independent
	identical blocks	instance-specific

motivated by engineering considerations, an alternative formulation of the rectangular block-packing problem in which all blocks need to fit into a predefined bounding box (fixed-outline) has been studied with increasing attention [1, 2, 4, 5, 7]. This constraint-satisfaction formulation enables hierarchical top-down block-packing which is useful for instances with thousands of blocks. The predefined bounding box facilitates more flexible notions of dominance. This is because we do not have to restrict packing U to have both sides not greater than those of packing U' when U dominates U', as long as U fits into the predefined bounding box.

References

1. S.N. Adya, I.L. Markov, "Fixed-outline Floorplanning: Enabling Hierarchical Design," to appear in *IEEE Trans. on VLSI*, 2003.
2. S.N. Adya, I.L. Markov, "Fixed-outline Floorplanning Through Better Local Search," *ICCD 2001*, pp. 328–334.
3. Y.-C. Chang et al., "B*-trees: A New Representation for Non-Slicing Floorplans," *DAC 2000*, pp. 458–463.
4. W. Choi and K. Bazargan "Hierarchical Global Floorplacement Using Simulated Annealing and Network Flow Area Migration", *DATE 2003*.
5. Y. Feng, D. P. Mehta, and H. Yang, "Constrained "Modern" Floorplanning", *ISPD 2003*, pp. 128-135.
6. P.-N. Guo, C.-K. Cheng, T. Yoshimura, "An O-tree Representation of Non-Slicing Floorplan and Its Applications," *DAC 1999*, pp. 268–273.
7. A. B. Kahng, "Classical Floorplanning Harmful?," *ISPD 2000*, pp. 207-213.
8. M. Lai and D. Wong, "Slicing Tree Is a Complete Floorplan Representation," *DATE 2001*, pp. 228–232.
9. H. Murata et al., "VLSI Module Placement Based on Rectangle-Packing by the Sequence-Pair," *IEEE Trans on CAD*, 15(12), pp. 1518–1524, 1996.
10. Y. Pang, C.-K Cheng, K.Lampaert and W.Xie, "Rectilinear Block Packing Using O-tree Representation," *ISPD 2001*, pp. 156–161.
11. S. Prestwitch, "Supersymmetric Modelling for Local Search", *SymCon '02*, September 2002; `http://user.it.uu.se/~pierref/astra/SymCon02/`.
12. D. F. Wong and C. L. Liu, "A New Algorithm for Floorplan Design," *DAC 1986*, pp. 101–107.
13. B. Yao et al., "Floorplan Representations: Complexity and Connections," *ACM Trans. on Design Autom. of Electronic Systems*, 8(1), pp. 55–80, 2003.
14. `http://www.cse.ucsc.edu/research/surf/GSRC/MCNCbench.html`