

# Are Floorplan Representations Important In Digital Design?

Hayward H. Chan<sup>†</sup>, Saurabh N. Adya<sup>‡</sup> and Igor L. Markov<sup>†</sup>

<sup>†</sup>The University of Michigan, Department of EECS, 1301 Beal Ave., Ann Arbor, MI 48109-2122

<sup>‡</sup>Synplicity Inc., 600 W. California Ave, Sunnyvale, CA 95054

{hhchan, imarkov}@umich.edu, saurabh@synplicity.com

## ABSTRACT

Research in floorplanning and block-packing has generated a variety of data structures to represent spatial configurations of circuit modules. Much of this work focuses on the geometry of module shapes and seeks tighter packing, as well as improvements in the asymptotic worst-case complexity of algorithms for standard tasks. In this work we consider the implications of interconnect optimization on the value of floorplan representations and establish a framework for comparing different representations. By analyzing performance bottlenecks in block packing and properties of floorplan representations, we show that many of the mathematical results in floorplanning do not translate into better VLSI layouts. This is confirmed by extensive empirical data for stand-alone floorplanners and integrated applications.

**Categories and Subject Descriptors:** B.7.2 [Integrated Circuits]: Design Aids — *placement and routing*; G.4 [Mathematical Software]: Algorithm Design and Analysis; J.6 [Computer-Aided Engineering]: Computer-Aided Design.

**General Terms:** Algorithms, experimentation.

**Keywords:** Circuit layout, floorplanning, sequence pair, B\*-tree.

## 1. INTRODUCTION

Floorplanning has traditionally been important in VLSI design because it determines the top-level spatial structure of a chip. Today automatic floorplanning is encouraged by the growing adoption of embedded memories and IP blocks in SoC designs. While human designers may have non-trivial insights into design objectives, they cannot analyze millions of possible spatial configurations.

A floorplan can be represented by the locations of the blocks, as in [10], but this complicates the generation of new overlap-free floorplans. We note that VLSI floorplanners typically rely on local search, especially simulated annealing, and spend most of their runtime to incrementally modify and evaluate candidate floorplans. Therefore, topological representations are more common as they guarantee that all encoded packings are overlap-free. Such a representation encodes relative positions among blocks in a way that is amenable to perturbation. For example, the sequence pair [16] captures a packing by a pair of permutations and can be modified in  $O(1)$  time, shifting the computational burden to the retrieval of block locations. Different representations are often compared by the algorithmic complexity of their *evaluation*.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

**Table 1: Published results of area-minimization on MCNC benchmarks of different representations, except for several results for CBL and ECBL that are incorrect (INC), according to the authors. Results known to be optimal are boldfaced.**

Floorplan representation	Published results				
	<i>apte</i>	<i>xerox</i>	<i>hp</i>	<i>ami33</i>	<i>ami49</i>
Optimal [4]	<b>46.9</b>	<b>19.8</b>	<b>8.95</b>	time-out	
Sequence pair [22]	<b>46.9</b>	<b>19.8</b>	<b>8.95</b>	1.21	36.8
TCG [13]	<b>46.9</b>	<b>19.8</b>	<b>8.95</b>	1.20	36.8
TCG-S [14]	<b>46.9</b>	<b>19.8</b>	<b>8.95</b>	1.19	36.4
O-tree [18]	<b>46.9</b>	20.2	9.16	1.24	37.7
B*-tree [5]	<b>46.9</b>	<b>19.8</b>	<b>8.95</b>	1.27	36.8
Corner-block list (CBL) [8]	INC	<b>19.8</b>	INC	1.18	36.1
ECBL [27]	INC	19.9	INC	1.19	36.7
Twin-binary sequence [26]	47.2	<b>19.8</b>	9.03	1.21	37.0
TBS extended [26]	47.4	<b>19.8</b>	9.02	1.19	36.9
Q-sequence [29]	<b>46.9</b>	19.9	9.03	1.19	36.8
ACG [28]	<b>46.9</b>	19.9	9.03	1.19	36.8

*Slicing* floorplans are those that can be recursively bisected by horizontal and vertical cut-lines down to single blocks. They can be encoded by slicing trees or Polish expressions [23]. The first encoding of an arbitrary floorplan — a sequence pair — appeared 10 years ago [16]. Today more than a dozen representations exist [5, 7, 8, 13, 14, 22, 26, 27, 28, 29]. Relations among them are studied in [25] and will be summarized in Section 2.

Competitive VLSI floorplanners frequently use simulated annealing because this allows modifying the objective function in applications. They differ in the choice of floorplan representation, which may affect runtime and solution quality. New floorplan representations are typically justified by improvements in the algorithmic complexity of evaluation, the type of encoded floorplans, the amount of redundancy in the encoding and the total number of encoded configurations. To this end, linear- or near-linear time evaluation algorithms are already known for a number of representations (sequence pair, B\*-tree, etc), and would be difficult to improve upon if the location of each block is required (a possible exception is incremental evaluation). Many existing representations encode a sufficiently broad range of floorplans.

Significant improvements are also unlikely because popular benchmarks are so small that many publications already report optimal solutions (see Table 1). The use of simulated annealing leads to particularly controversial reporting of results. For example, many papers [5, 13, 14, 29] only report best results out of an unknown number of independent runs. Since temperature schedules are rarely reported, it is conceivable to tune them to individual benchmarks. Needless to say, such results may be difficult to reproduce. For example, the performance of TCG and TCG-S is questioned in [28] since there are significant discrepancies between the experimental results and results reported in [13, 14]. Many rep-

representations [5, 22, 28, 29] have not been evaluated in the context of interconnect optimization, and in other cases, wirelength optimization is shown in a token experiment without much discussion. For example, ACG in [28] is presented as “more suitable for interconnect plan” than existing representations, but no results on interconnect are reported. This state of confusion reminds of the critical analysis of VLSI placement literature in [15], and our work is motivated similarly. However, a major difference is that many results in floorplanning have been mathematically proven, therefore our focus is on relevance and applicability rather than correctness.

Larger-scale layout is a good context for justifying improvements in asymptotic complexity of optimization algorithms, and excellent area-packing results were reported recently for benchmarks with over 100 blocks [4, 12]. However, these results have little or no impact on interconnect minimization. The work in [2] suggests that in fixed-outline interconnect minimization [9], annealing is outperformed for over 100 blocks by a hybrid algorithm that uses min-cut partitioning where possible and resorts to annealing-based packing when necessary. Whether or not the choice of topological floorplan representations is significant for interconnect-driven floorplanning is one of the questions addressed in our work.

In this work we group existing floorplan representations into families of related data structures that have identical “solution spaces” and/or share similar evaluation algorithms. To avoid redundant comparisons, we then select (i) sequence pair [16] that is used in the well-known tool Parquet [1] and is related to TCG and TCG-S [14], as well as (ii) B\*-tree [24], which is similar to O-tree [18] and for which a multi-level extension has been reported [12]. Several unrelated representations (e.g., mosaic floorplans) are excluded from our comparison because their applicability is more restricted. We replace sequence pair with B\*-tree in the annealing-based floorplanner Parquet [1] so as to compare the two representations using exact same temperature schedule. Additionally, making fair and realistic comparisons requires several technical results, which we develop in this work. Empirical data show that the size of the solution space of a representation and worst-case complexity of its evaluation are not very relevant to actual performance. To scale our analysis to mixed-size placement, we embed floorplanning comparisons into the framework of *min-cut floorplacement* that additionally involves min-cut partitioning [2] and has been demonstrated on up to 220K movable objects. In this framework, the layout is first partitioned into subregions, minimizing the interconnect between them. Fixed-outline floorplanning is invoked in regions with relatively large macros in it. In our work floorplacement is used for benchmarking purposes as a practical application of floorplanning where both packing and interconnect optimization are significant. In terms of stand-alone evaluation of block packers, floorplacement generates a large number of diverse floorplanning instances allowing an unbiased comparison of different block packing techniques.

In the remaining part of the paper, Section 2 describes and compares existing floorplan representations. We outline our evaluation framework in Section 3, present and analyze experimental results in Section 4 and conclude with a discussion in Section 5.

## 2. FLOORPLAN REPRESENTATIONS

A key property of popular topological floorplan representations is that they capture at least one area-optimal solution. Other than that, their solution spaces may be quite different. The floorplan in Figure 3c cannot be captured by a B\*-tree but can be captured by a sequence pair [16] and by a corner-block list [8]. Some representations may actually capture the exact same set of floorplans, such as sequence pair, TCG and TCG-S. Therefore, we distinguish several families of topological representations.

### 2.1 Families of representations

Sequence pair [16], TCG and TCG-S [14] are shown to be equivalent in [14] in the sense that they share the same  $(n!)^2$  solution space and capture the exact same set of floorplans. Each sequence pair corresponds to one TCG and vice versa. TCG-S is a hybrid of TCG and sequence pair, which contains a constraint graph from TCG and a sequence from sequence pair. Since TCG and TCG-S capture the same set of floorplans and take  $O(n^2)$  time to evaluate, we only evaluate sequence pair since (a) it is simpler, and (b) annealing moves takes less time to evaluate. Multiple sequence pairs can represent the same floorplan, as we will see in Section 2.2, and the exact number of general floorplans is in  $O(n!2^{5n}/n^{4.5})$ , which is much smaller than  $(n!)^2$  [20]. This redundancy is typically viewed as a limitation of sequence pair, TCG and TCG-S. However, such an argument is only applicable to exhaustive-search algorithms (as in [4]), but not necessarily local search. The literature on Boolean Satisfiability suggests that redundancies in the solution space, e.g., symmetries [19], make local search more successful by increasing the number of paths leading to desirable configurations (i.e., increasing the basins of attraction of global optima).

Both O-tree [7] and B\*-tree [5] use a single tree to represent a horizontally compacted packing (Fig.2), but differ in the bit-level implementation of the tree. O-tree uses a rooted-ordered tree with arbitrary vertex degrees, while B\*-tree uses a binary tree. Therefore they share the same  $O(n!2^{2n-2}/n^{1.5})$  solution space and capture the same set of floorplans. As pointed out in [5], the regularity of B\*-tree makes it more attractive in annealing-based floorplanners.

Another family of floorplan representations captures only floorplans with zero wasted area (*mosaic floorplans*) [8] and includes the corner-block list, twin-binary sequence [25] and Q-sequence [29]. In practical applications with fixed-sized blocks (e.g., embedded memories, datapaths and IP blocks) it is extremely rare that no space is wasted. Therefore some mosaic representations are extended to handle general floorplans by inserting empty rooms. Area-optimality is guaranteed when enough rooms are inserted. ECBL [27], an extension of corner-block list, requires  $n^2$  empty rooms to capture the area-optimal solutions, but this leads to a dramatic growth of the solution space and makes evaluation runtimes impractical. Extensions of twin binary sequence to capture area-optimal floorplans without redundancies [26] do not outperform published results for sequence pair in [22]. Mathematical properties of various floorplan representations are discussed in [25].

If the representations share the same solution space, one can equalize the move set, and thus the differences will only affect runtime (some moves may be faster or slower). Therefore, we do not consider TCG and extrapolate results for sequence pair to TCG. Similarly, we study B\*-trees and extrapolate results to O-trees, with a potential caveat about runtime. On the other hand, the solution space defines an intrinsic bound on the expressiveness of a representation, and this bound may directly impact solution quality. The example in Figure 3c shows that B\*-tree may not necessarily capture min-wirelength solutions that can be captured by sequence pair. Therefore, we compare solution spaces rather than specific details of individual floorplan representations.

Incidentally, the two representations we have chosen for comparisons, sequence pair and B\*-tree, appear best-studied in the literature among non-slicing representations. Sequence pair has been extended to handle fixed blocks [17], arbitrary convex and concave rectilinear blocks [6] and soft blocks [1]. Similar extensions have been proposed for B\*-tree [24] or can be easily extrapolated. Both representations have been used in multi-level or hierarchical floorplanning [1, 12] scaling to tens of thousand blocks.

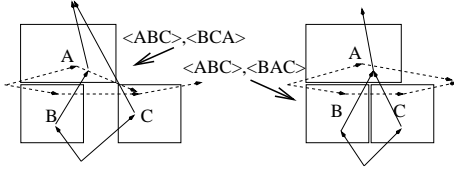
## 2.2 Sequence pair

Unlike graph-based representations, a sequence pair [16] is a pair of permutations (orderings) of the  $N$  blocks. The two permutations capture geometric relations between each two blocks. Recall that since blocks cannot overlap, one of them must be to the left or below from the other, or both. In sequence pair

$$\langle \dots, a, \dots, b, \dots \rangle, \langle \dots, a, \dots, b, \dots \rangle \Rightarrow a \text{ is to the left of } b \quad (1)$$

$$\langle \dots, a, \dots, b, \dots \rangle, \langle \dots, b, \dots, a, \dots \rangle \Rightarrow a \text{ is above } b \quad (2)$$

Every two blocks constrain each other in either vertical or horizontal direction, and only these constraints are recorded. Therefore, placements produced from sequence pair must be aligned to given horizontal and vertical axes, e.g.,  $x = 0$  and  $y = 0$ . There are  $n!^2$  sequence pairs for  $n$  blocks, but multiple sequence pairs may encode the same block placement, e.g., for three identical square blocks, both  $\langle a, c, b \rangle, \langle c, a, b \rangle$  and  $\langle a, c, b \rangle, \langle c, b, a \rangle$  encode the placement with  $a$  straight on top of  $c$ , and  $b$  aligned with  $c$  on the right (Fig.3a). We say that a block placement is “representable” (or “can be captured”) by a sequence pair *iff* there exists a sequence pair which encodes that placement. The use of sequence pair for area minimization is justified by the fact [16] that at least one minimal-area placement is representable.



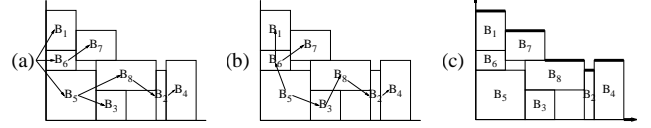
**Figure 1: Two sequence pairs with edges of the horizontal (dashed) and vertical (solid) constraint graphs. Transitive edges are omitted.**

The original  $O(n^2)$ -time evaluation algorithm from [16] has been considerably simplified in [21]. Another variant in [21] runs in time  $O(n \log(n))$ , and later work in [22] reduces runtime to  $O(n \log(\log(n)))$  without affecting the resulting block locations. While O-trees [18] and corner block lists [8] can be evaluated in linear time, the difference in complexity is dwarfed by implementation tuning, e.g., the annealing schedule. The implementation in [22] outperforms many published results in terms of runtime and solution quality.

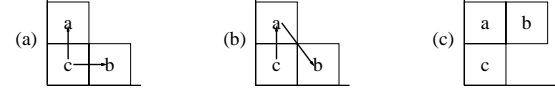
## 2.3 B\*-tree

B\*-tree represents a compacted packing by a binary tree, in which each node corresponds to a block (see Fig.2b). The root node represents the bottom-left block, which must exist if we compactify in that direction. For example,  $B_5$  in Fig.2b is the bottom-left block. A left child is the lowest right neighbor of its parent and a right child is the lowest block above its parent that shares the same  $x$ -coordinate with its parent. In Fig.2b,  $B_7$  and  $B_1$  are the left and right children of  $B_6$  respectively.

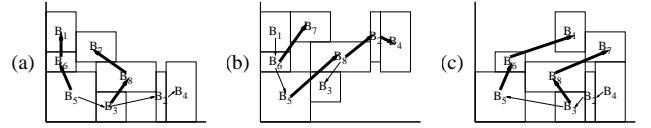
Given a B\*-tree, block locations can be found by a depth-first traversal of the tree. After block  $A$  is placed at  $(x_A, y_A)$ , we consider its left child  $B$  and set  $x_B = x_A + w_A$ , where  $w_A$  is the width of  $A$ . Then  $y_B$  is the smallest non-negative value that avoids overlaps with previously placed blocks. After returning from recursion at block  $B$ , we consider the right child  $C$  of  $A$ :  $x_C = x_B$ , and  $y_C$  is smallest possible so as to avoid overlaps. This algorithm can be implemented in  $O(n)$  time with the *contour* data-structure. The contour of a packing defines its upper outline (jagged) and can be implemented as a doubly-linked list of line segments (Fig.2c). When we put a new block on top of the contour at a certain  $x$ -coordinate, it takes amortized  $O(1)$  time to determine its  $y$ -coordinate [5].



**Figure 2: A packing represented by (a) an O-tree and (b) an equivalent B\*-tree; the contour of the packing is shown in (c). Block  $B_5$  is the root node. Thicker arrows link parents to their left children.**



**Figure 3: (a)-(b) Multiple B\*-trees can represent the same packing; B\*-tree does not capture packing (c), which is represented by the sequence pair  $\langle a, b, c \rangle, \langle c, a, b \rangle$ .**



**Figure 4: (a) shows the vertical B\*-tree of the horizontal B\*-tree in Fig.2b. (b) shows the reverse horizontal B\*-tree of the B\*-tree in Fig.2b. All blocks are packed to the top instead of the bottom. (c) shows the reverse vertical B\*-tree of the vertical B\*-tree in (a). All blocks are packed to the right. Blocks are linked with their left-child by boldfaced arrows.**

All packings represented by B\*-trees are necessarily compacted so that no single block can move down without creating overlaps. Therefore, some packings representable by sequence pair cannot be captured by B\*-tree, as shown in Fig.3c. The same packing can be captured by multiple B\*-trees (Fig.3). There are  $O(n!2^{2n-2}/n^{1.5})$  B\*-trees [5] packings, which is fewer than  $n!^2$  possible sequence pair. B\*-tree captures all compacted packings, and hence some area-optimal packings. However, it may capture none of interconnect-optimal packings.

## 3. OUR EVALUATION FRAMEWORK

Our evaluation framework is based on the open-source floorplanner Parquet-2.1 that implements simulated annealing and uses the sequence pair representation [1]. The temperature schedule can be easily extracted from its source code. Parquet supports both traditional outline-free min-area floorplanning and fixed-outline floorplanning [9] for any combination of hard and soft blocks, with optional interconnect optimization. We replace sequence pair with B\*-tree in Parquet to facilitate fair comparisons with exact same temperature schedules. With some effort, we ensure that all features of Parquet are supported when B\*-tree is used. Probabilities of trying and accepting different moves are as similar as possible, although the move sets for the two representations are slightly different. All our implementations are available in Parquet 3.

### 3.1 Floorplanning

**Outline-free floorplanning.** The following moves are used with B\*-tree, (i) swapping two blocks, (ii) rotating a block and (iii) moving a block to another position in the tree (specific to B\*-tree). The following moves are used with sequence pair: (a) swapping two blocks in either sequence or both, (b) rotating a block and (c) deleting a block from one of the sequences and inserting it at a random position (specific to sequence pair). During interconnect

optimization, movements of single blocks are guided towards locations that optimize adjacent interconnect. For the two floorplan representations, block rotations and interconnect-driven moves are applied with the same probabilities. To investigate the effects of compaction on both area and interconnect optimization, we apply the compaction algorithm in Section 2.3 every five moves, and report floorplanning results with and without this feature.

**Floorplan compaction for B\*-tree.** Compaction has been studied in [7, 11] with the goal to compress a given floorplan in both directions. Note that packings captured by B\*-trees and O-trees are only guaranteed to be compacted in one direction (down). We developed the following simple compaction scheme. Given a (horizontal) B\*-tree, we construct a vertical B\*-tree that compacts all blocks to the left. For example the horizontal B\*-tree in Fig.2b has vertical B\*-tree in Fig.4a. From a vertical B\*-tree, we construct a horizontal B\*-tree and then iterate this process until all blocks are compacted downward and to the left. Our algorithm for “re-orienting” a B\*-tree is derived from the B\*-tree evaluation algorithm, which places blocks one by one, in a depth-first order. When we place a block  $B$ , we identify the block  $A$  (or the bottom edge) that prevents it from moving towards the bottom as its parent, and try to mark it as the left-child of  $A$ . If block  $A$  already has left-child  $C$ , we try to mark the new block  $B$  as the right-child of  $C$ . If  $C$  already has a right-child, we consider the right-child of  $C$ , proceed similarly until we reach a block  $C'$  with no right-child and mark new block  $B$  as the right-child of  $C'$ . In Fig.4a, for example, when we place the block  $B_2$ , we identify the bottom edge as its parent, whose left-child is already set to be block  $B_5$ . Therefore, we consider its right-child  $B_3$ , which has no right-child, and set  $B_2$  as its right-child. On the other hand, when block  $B_7$  is placed, we identify block  $B_8$  as its parent. Since block  $B_8$  has no left-child, we set  $B_7$  to be its left-child. B\*-tree evaluation takes  $O(n)$  time for  $n$  blocks, but the worst-case overhead for placing block  $B$  in our re-orientation algorithm is  $O(n)$  due to the tree-traversal needed to find  $B$ 's parent. We can avoid this traversal by recording the last left-child of each block, so that next time when we add a block  $B$  as a left-child of block  $A$ , we can go directly to the last left-child of  $A$ . For example in Fig.4a, when we place the block  $B_2$ , we read from the bottom-edge that  $B_3$  is the last left-child of the bottom-edge. Therefore, we can consider  $B_3$  directly, and add  $B_2$  as the left-child of  $B_3$  without tree traversal. After we place the new block  $B_2$ , we update  $B_2$  as the last left-child of the bottom-edge. This last optimization enables constructing the vertical B\*-tree in  $O(n)$  time.

**Fixed-outline floorplanning.** In the fixed-outline mode [9], Parquet uses slack-based moves from [1]. The  $x$ -slack of a block is the distance by which it can move with other blocks fixed. A key result in [1] shows that the width of a floorplan cannot be shortened unless some block with zero  $x$ -slack is moved, and similarly for  $y$ -slacks. The slack computation in [1] is specific to sequence pair, and in this work we develop slack computation for B\*-tree. Conceptually, we construct a *reverse* B\*-tree for a horizontal B\*-tree that compacts blocks to the top, instead of the bottom (Fig.4b). The difference in  $y$ -coordinates of each block in these two horizontal B\*-tree is its  $y$ -slack. To evaluate the  $x$ -slacks, we first construct the vertical B\*-tree  $T$  that compacts blocks to the left. Then we construct the reverse vertical B\*-tree  $T'$  from  $T$  that compacts blocks to the right (Fig.4c). The difference in  $x$ -coordinates of each block in  $T'$  and the original horizontal B\*-tree is its  $x$ -slack. Note that slack computation for B\*-tree takes  $O(n)$  time. This allows us to make slack-based moves identical for B\*-tree and sequence pair.

**Optimizing soft blocks.** Aspect ratios of soft blocks can be optimized based on their  $x$  and  $y$  slacks. For example, we reshape blocks with zero  $x$ -slack and positive  $y$ -slack by decreasing their

widths and increasing their heights. This reduces the number of critical paths in the  $x$  direction without creating new critical paths in the  $y$  direction, and may improve floorplan area. Again, we ensure that the soft-block moves for sequence pair and B\*-tree are as similar as possible, and applied with the same probability. Empirical data suggest the same trends for floorplanning with hard blocks and for floorplanning with soft blocks. Therefore, we mainly report on hard-block floorplanning and illustrate the similarity by representative results with soft blocks.

## 3.2 Floorplacement

Min-cut floorplacement introduced in [2] integrates fixed-outline floorplanning into traditional min-cut placement to solve a more general layout problem, which includes cell placement, floorplanning, mixed-size placement and achieving routability. At every step of min-cut placement, either partitioning or wirelength-driven, fixed-outline floorplanning is invoked, depending on whether large blocks are present in a given layout region. If floorplanning fails to satisfy the fixed outline, an earlier partitioning decision is undone, adjacent regions are merged and the larger region is re-floorplanned to find a legal placement of the blocks. Empirically, this framework improves the scalability and quality of results for traditional wirelength-driven floorplanning. A major implication of min-cut floorplacement is that large layouts with blocks of different sizes may not require large-scale block packing. Indeed, min-cut partitioning may divide the core regions into smaller sub-region before block packing is involved. For a given layout region (bin), a block-packing instance is constructed as follows. All connections between modules in the bin and other modules are propagated to *fixed terminals* at the periphery of the bin. As the bin may contain numerous standard cells, the number of movable objects is reduced by clustering standard cells into soft placeable blocks using a simple bottom-up connectivity-based algorithm. Our implementation only clusters small cells but not large modules.

Preliminary results of floorplacement experiments reported in the next section indicated to us that wirelength evaluation in block packing consumed a significant portion of runtime. Therefore we made several modifications to the floorplacer described in [2], so as to reduce the number of nets in block-packing instances it generates. First, when formulating a block-packing instance, the modified floorplacer ignores all nets whose bounding boxes contain the fixed outline of the instance. Such nets do not affect the optimality of block-packing solutions, and removing them is a clear win. This is similar to removing *inessential nets* in min-cut placement [3]. The main difference is that min-cut partitioning affects only the  $x$ -span or the  $y$ -span of each net, depending on the direction of the cutline, but floorplanning affects both. Another improvement to the floorplacer from [2] deals with soft blocks that consist of small cells clustered together. We noticed that in many cases a pair of such clusters would be connected by numerous two-pin nets that can be conglomerated. Therefore, we (i) extended the Parquet floorplanner used in [2] to use net weights, and (ii) developed a near-linear-time algorithm for conglomerating “parallel” two-pin nets. This algorithm creates an adjacency vector (expandable array) for every vertex, in which it stores indices of other vertices adjacent to the current vertex through two-pin nets. Each adjacency vector is sorted, after which repeated indices always appear next to each other and can be counted in linear time. Each group of identical indices is then represented by a single two-pin net whose weight is the multiplicity of the index. Empirically, these two improvements significantly reduce the number of nets considered during block packing, resulting in an overall 10% speed-up of min-cut floorplacement on the 18 IBM benchmarks (more so on larger benchmarks) without a noticeable impact on solution quality.

**Table 2: Comparison of sequence pair and B\*-tree in outline-free mode with hard blocks. Parquet is run on a 3.2GHz Linux workstation with 1GB RAM, and all data are averaged over 50 independent runs.**

		area weight = 1.0		area weight = 0.6, wire weight = 0.4		time-per-move (area_only : area+wire)	
		dead-space % / HPWL (e3) / runtime (s) / time-per-move (ms)				estimated	actual
ami33	sequence pair	10.4%	133 / 0.25s / 5ms	14.3%	75 / 1.38s / 30ms	1 : 8.28	1 : 6.00
	B*-tree (no compaction)	5.72%	145 / 0.33s / 7ms	13.5%	76 / 1.19s / 30ms		1 : 4.29
	B*-tree (with compaction)	5.54%	140 / 0.32s / 7ms	13.4%	75 / 1.17s / 30ms		1 : 4.29
ami49	sequence pair	9.19%	1853 / 0.53s / 8ms	14.8%	849 / 3.31s / 50ms	1 : 7.19	1 : 6.25
	B*-tree (no compaction)	5.37%	2177 / 0.72s / 10ms	14.9%	856 / 3.40s / 59ms		1 : 5.90
	B*-tree (with compaction)	5.05%	2074 / 0.68s / 10ms	14.5%	893 / 2.89s / 51ms		1 : 5.10
n100	sequence pair	9.79%	399 / 2.38s / 18ms	11.9%	325 / 27.8s / 215ms	1 : 5.48	1 : 11.9
	B*-tree (no compaction)	5.50%	460 / 2.82s / 19ms	8.33%	330 / 26.7s / 233ms		1 : 12.3
	B*-tree (with compaction)	5.36%	481 / 2.73s / 19ms	8.35%	330 / 26.4s / 233ms		1 : 12.3
n200	sequence pair	11.0%	735 / 10.6s / 41ms	11.2%	576 / 150s / 580ms	1 : 3.95	1 : 14.1
	B*-tree (no compaction)	6.18%	846 / 11.4s / 39ms	9.23%	602 / 136s / 604ms		1 : 15.5
	B*-tree (with compaction)	6.16%	851 / 11.0s / 37ms	9.34%	605 / 136s / 604ms		1 : 16.3
n300	sequence pair	11.4%	958 / 26.4s / 70ms	15.4%	716 / 297s / 759ms	1 : 3.18	1 : 10.9
	B*-tree (no compaction)	6.65%	1139 / 25.6s / 58ms	9.51%	745 / 268s / 783ms		1 : 13.5
	B*-tree (with compaction)	6.57%	1217 / 24.6s / 55ms	9.56%	745 / 269s / 784ms		1 : 14.3

## 4. EMPIRICAL RESULTS AND ANALYSIS

We now compare sequence pair to B\*-tree within simulated annealing in several different contexts. All experiments are performed on a 3.2GHz Pentium4 workstation with 1GB RAM running Linux.

### 4.1 Floorplanning

The first batch of results is on the MCNC and GSRC benchmarks (see Table 3), except for those with 11 blocks and smaller.

**Outline-free floorplanning.** In Table 2 we report average results over 50 runs, while best-seen dead-space is typically 1-2% smaller. Evidently B\*-tree packs better than sequence pair, which is consistent with B\*-tree’s capturing only horizontally compacted floorplans and with published results [5]. By profiling average time per move, we empirically determine that floorplan evaluation dominates in pure area-optimization mode, i.e., when the netlist is not considered. This is consistent with the emphasis on algorithm complexity in the floorplanning literature. For example, the linear-time algorithm for evaluating a B\*-tree is considered faster than the naive  $O(n^2)$  algorithm for sequence pair. However, as we see in Table 2, this asymptotic comparison has little value for GSRC and MCNC benchmarks with 200 blocks or less. The more sophisticated B\*-tree evaluation is faster than sequence pair evaluation *only with over 200 blocks*. To this end, we recall that  $O(n \log(n))$ -time and  $O(n \log(\log(n)))$ -time evaluation algorithms exist for sequence pair. In another surprising result, repeated compaction of the floorplan does not lead to significantly better results even when only area is minimized, which suggests that floorplan compaction algorithms are only useful in post-processing.

**Interconnect optimization.** When we configure our annealer to optimize a linear combination of area and wirelength, B\*-tree leads to smaller packings, while sequence pair achieves smaller wirelength (see Table 2). This is consistent with the observation that B\*-tree does not capture interconnect-optimized packings that are not horizontally compacted. Table 2 shows that runtime is now dominated by wirelength evaluation, as time-per-move quadruples. Intuitively this makes sense because floorplanning instances usually have many more nets and pins than blocks (see Table 3). In this

context floorplan evaluation is not a runtime bottleneck in practice. Below, we compare the number of floating-point operations used by wirelength computation and that by floorplan evaluation, to show that *floorplan evaluation should not be a bottleneck in general*.

First, we optimistically count floating-point operations needed to compute the HPWL of a netlist for a given placement (we ignore assignments, etc). The location of each pin  $p$  is given by the location  $(b_x, b_y)$  of its module’s center, module dimensions  $(w, h)$  and the pin’s relative offset  $(f_x, f_y)$  from the center. Namely,  $p_x = b_x + wf_x$  and similarly for  $p_y$  (here  $wf_x$  can be precomputed for hard blocks). Thus, each pin requires 4 operations to compute its location, and absolute locations of pads are given. To calculate the HPWL of a degree- $d$  net, one finds the smallest and largest  $x$  and  $y$  pin coordinates, which takes  $4d$  floating-point comparisons. A special-case computation for 2-pin nets uses only two comparisons, saving 6 comparisons out of 8. Three more operations compute the HPWL of a net as  $(x_{max} - x_{min}) + (y_{max} - y_{min})$ . Now assume that the netlist has  $p$  pins,  $q$  pads and  $N$  nets of which  $N_2$  are two-pin nets and observe that  $p + q = \sum_i d_i$ . Hence the number of floating-point operations to compute the HPWL is  $4p + 4(p + q) + 3N - 6N_2$ .

Next, we consider the original  $O(n^2)$ -time algorithm for evaluating sequence pairs [16]. It uses the horizontal and vertical constraint graphs with  $E_H$  and  $E_V$  edges respectively. Block locations are calculated by two depth-first traversals, which requires  $(E_H + E_V)$  floating-point additions and as many comparisons. Each pair of blocks is either horizontally or vertically constrained, and each block is connected to the sources and sinks of the two constraint graphs (Fig.1). Therefore  $E_H + E_V = \binom{n}{2} + 4n$ , and the total is  $2(E_H + E_V) = n(n + 7)$  floating-point operations. Table 3 estimates such operation counts for evaluating sequence pair and HPWL, per benchmark. It also counts actual floating-point operations in the faster algorithm that we use [21] (which does not build constraint graphs). Both algorithms have worst-case complexity  $O(n^2)$ , and the one from [16] always takes the same amount of time. However, plotting the data in Table 3 suggests a difference in asymptotic average-case behavior. Using more than 500 randomly-generated floorplans ranging from 16 to 16K blocks, we fit the performance of the faster algorithm to  $cn^{1.3}$  with good accuracy. Further, the overhead of wirelength evaluation is shown in Table 2. Surprisingly, the actual overhead increases for larger benchmarks, contrary to what floating-point counts suggest. To this end, we observe that wirelength evaluation operates on much larger datasets than floorplan evaluation (see Table 3), and larger netlists may not fit in processor cache. Overall, floorplan evaluation is never a runtime bottleneck in our experiments.

**Table 3: Attributes of MCNC and GSRC benchmarks**

	# blks	# nets	# pads	# pins	# fbat ops: estimated actual		
					WL eval.	seq. pair eval.	
ami33	33	123	42	522	4089	1320	562
ami49	49	408	22	953	6914	2744	1117
n100	100	885	334	1873	14253	10700	3180
n200	200	1585	564	3599	28669	41400	9710
n300	300	1893	569	4358	34593	92100	15863

**Table 4: Comparison of sequence pair and B\*-tree in fixed-outline mode without HPWL optimization. Parquet is run on a 3.2GHz Linux workstation with 1GB RAM, and all data are averaged over 50 independent runs. All blocks are hard.**

	time per move	10% dead-space		20% dead-space		
		aspect ratio = 1.0	aspect ratio = 2.0	aspect ratio = 1.0	aspect ratio = 2.0	
		success-rate % / average runtime				
<i>ami33</i>	sequence pair	12ms	32% / 0.39s	40% / 0.38s	100% / 0.13s	40% / 0.04s
	B*-tree (no compaction)	18ms	88% / 0.27s	82% / 0.34s	100% / 0.07s	100% / 0.11s
	B*-tree (with compaction)	18ms	92% / 0.27s	86% / 0.33s	100% / 0.07s	100% / 0.10s
<i>ami49</i>	sequence pair	18ms	70% / 0.70s	74% / 0.74s	100% / 0.25s	100% / 0.26s
	B*-tree (no compaction)	26ms	86% / 0.66s	98% / 0.63s	100% / 0.19s	94% / 0.24s
	B*-tree (with compaction)	24ms	82% / 0.70s	86% / 0.72s	100% / 0.07s	100% / 0.10s
<i>n100</i>	sequence pair	43ms	68% / 3.73s	52% / 4.00s	100% / 1.41s	100% / 1.36s
	B*-tree (no compaction)	54ms	100% / 2.47s	100% / 2.52s	100% / 1.03s	100% / 1.10s
	B*-tree (with compaction)	52ms	98% / 2.48s	100% / 2.42s	100% / 1.04s	100% / 1.04s
<i>n200</i>	sequence pair	150ms	56% / 17.7s	22% / 19.6s	100% / 6.82s	100% / 7.01s
	B*-tree (no compaction)	174ms	94% / 12.6%	100% / 13.3s	100% / 5.19s	100% / 5.20s
	B*-tree (with compaction)	170ms	100% / 12.1s	100% / 12.3s	100% / 5.24s	98% / 5.22s
<i>n300</i>	sequence pair	337ms	34% / 43.3s	10% / 48.7s	100% / 18.2s	100% / 18.9s
	B*-tree (no compaction)	414ms	98% / 29.1s	100% / 32.4s	100% / 13.0s	98% / 13.1s
	B*-tree (with compaction)	400ms	100% / 29.0s	100% / 29.6s	100% / 12.7s	100% / 12.8s

**Fixed-outline floorplanning.** Tables 4 and 5 show that moves are more expensive in fixed-outline floorplanning. In particular, the average time-per-move for B\*-tree grows faster than what its linear-time evaluation algorithm suggests. We trace these effects to slack-based moves that involve computing the  $x$ - and  $y$ -slacks for all blocks, and then prioritizing blocks by slacks. For both representations, this involves two floorplan evaluations, two rounds of floating-point subtractions, etc. However, slack-based moves are important to ensure that the final floorplan fits into a given outline [1]. B\*-tree typically achieves higher success rates than sequence pair, which is expected since B\*-tree captures only horizontally-compacted packings. Since Parquet terminates soon after finding the first legal solution, higher success rates improve runtime.

When optimizing interconnect subject to a fixed-outline, one is trading off success rate and interconnect cost. More aggressive annealers explore solutions with smaller wirelength at the risk of not finding any legal floorplans that fit into the outline. Therefore, success rates in Table 5 are substantially lower than those in Table 4. Due to compaction, B\*-tree packings are more likely to fit into the outline than those represented by sequence pair. Therefore one can view B\*-tree as more conservative and may expect to achieve the same trade-off by increasing or decreasing the weight of the interconnect term in the objective function of simulated annealing. However, *some low-interconnect floorplans may elude B\*-tree with any and all weight configurations*. As in the outline-free mode, wirelength evaluation dominates runtime, and periodic floorplan compaction does not improve results.

**Optimizing soft blocks.** The trends we reported so far carry over to soft blocks — Table 6 compares sequence pair and B\*-tree on soft-block version of MCNC and GSRC benchmarks, with compaction moves disabled. As before, (i) B\*-tree achieves higher success rates and takes less time on average, and (ii) wirelength evaluation considerably increases average time per move.

## 4.2 Floorplacement

To scale our previous comparisons up, we embed them into the min-cut floorplacement framework, where (i) the netlist and the layout area are first partitioned into smaller regions, and (ii) block packing is used only when macros occupy a significant portion of region. We work with the floorplacer Capo 9.0 [2], which uses the block packer Parquet. With minor modifications we can now use either sequence pair or B\*-tree within Capo. Our experiments are performed with a suite of 18 publicly-available mixed-size placement benchmarks *IBM-MSwPins* [2] that have non-square blocks

**Table 6: Comparison of sequence pair and B\*-tree in fixed-outline mode. All data are averaged over 50 independent runs, and all blocks are soft. The outline has 20% whitespace and aspect ratio 1.**

	Without HPWL optimization			
	sequence pair		B*-tree	
	success rate %	runtime (s)	success rate %	time-per-move (ms)
<i>ami33</i>	100%	0.14s / 11ms	100%	0.08s / 18ms
<i>ami49</i>	100%	0.34s / 18ms	100%	0.16s / 26ms
<i>n100</i>	100%	1.72s / 40ms	100%	1.26s / 56ms
<i>n200</i>	100%	10.1s / 155ms	100%	5.48s / 183ms
<i>n300</i>	100%	29.8s / 346ms	100%	13.9s / 431ms

	With HPWL optimization			
	sequence pair		B*-tree	
	success rate % / HPWL (e3)	runtime (s)	success rate %	time-per-move (ms)
<i>ami33</i>	82% / 83	1.27s / 37ms	90% / 85	0.85s / 43ms
<i>ami49</i>	88% / 926	3.16s / 62ms	88% / 1061	2.26s / 71ms
<i>n100</i>	100% / 344	23.6s / 241ms	100% / 355	13.9s / 286ms
<i>n200</i>	100% / 626	132s / 696ms	100% / 666	71s / 779ms
<i>n300</i>	100% / 759	277s / 986ms	100% / 843	142s / 1167ms

and non-trivial pin offsets. Final wirelengths and overall runtimes of mixed-size placement are shown in Table 7. Sequence pair outperforms B\*-tree by <2.5% in wirelength and, marginally, in runtime.<sup>1</sup> The difference is insufficient to declare a winner as further tuning may improve results by 1-2%. However, the choice of floorplan representation appears to make little impact overall. Of course, this is partly due to the fact that final results of mixed-size placement also depend on other subsystems of Capo, such as several min-cut partitioners and end-case placers. We also note that regularly compacting B\*-tree floorplans during annealing does not impact final results. The fraction of floorplacement runtime used by block packing is also reported, where we again see little difference.

Our next experiment aims to distinguish block-packing effects from floorplacement, and yet avoid overspecializing to MCNC and GSRC benchmarks. We collect a rich set of over 1300 block-packing instances generated during floorplacement and solve them with *stand-alone* sequence pair and B\*-tree floorplanners. Recall that during floorplacement standard cells may be clustered into soft blocks, as described in Section 3.2. Table 8 suggests that our saved instances are fairly diverse, ranging from 1 or 2 blocks to more than 100 blocks, which is interesting for benchmarking purposes. While average block counts are small, fairly large block-packing

<sup>1</sup>In min-cut placement, it is common to see that better optimization of interconnect at higher levels speeds up lower levels.

**Table 5: Comparison of sequence pair and B\*-tree in fixed-outline mode with HPWL optimization. Parquet is run on a 3.2GHz Linux workstation with 1GB RAM, and all data are averaged over 50 independent runs. All blocks are hard.**

	time per move	10% dead-space		20% dead-space		
		aspect ratio = 1.0	aspect ratio = 2.0	aspect ratio = 1.0	aspect ratio = 2.0	
		success-rate % / HPWL / average runtime				
ami33	sequence pair	38ms	8% / 82 / 1.60s	2% / 88 / 1.62s	82% / 81 / 1.26s	84% / 83 / 1.25s
	B*-tree (no compaction)	43ms	8% / 81 / 1.55s	0% / - / 1.58s	94% / 84 / 0.81a	96% / 84 / 0.75s
	B*-tree (with compaction)	42ms	2% / 84 / 1.54s	0% / - / 1.55s	90% / 83 / 0.78s	90% / 84 / 0.86s
ami49	sequence pair	76ms	2% / 1054 / 4.06s	0% / - / 4.30s	96% / 959 / 2.93s	82% / 989 / 3.74s
	B*-tree (no compaction)	75ms	0% / - / 4.10s	0% / - / 4.09s	82% / 1067 / 1.95s	86% / 1123 / 2.05s
	B*-tree (with compaction)	74ms	6% / 1052 / 3.94s	2% / 1245 / 3.74s	88% / 1069 / 2.17s	84% / 1116 / 2.32s
n100	sequence pair	251ms	20% / 345 / 30.5s	26% / 354 / 31.4s	100% / 342 / 23.6s	100% / 350 / 22.9s
	B*-tree (no compaction)	295ms	62% / 348 / 23.1s	32% / 353 / 27.3s	100% / 355 / 13.8s	100% / 362 / 13.6s
	B*-tree (with compaction)	286ms	48% / 345 / 25.7s	30% / 350 / 27.3s	100% / 355 / 13.4s	100% / 360 / 14s
n200	sequence pair	715ms	12% / 625 / 165s	2% / 660 / 168s	100% / 623 / 122s	100% / 643 / 123s
	B*-tree (no compaction)	772ms	28% / 650 / 135s	6% / 657 / 150s	100% / 663 / 67.7s	100% / 678 / 68.1s
	B*-tree (with compaction)	779ms	18% / 643 / 144s	6% / 670 / 148s	100% / 662 / 67s	100% / 679 / 68s
n300	sequence pair	1023ms	8% / 763 / 333s	0% / - / 339s	100% / 760 / 243s	100% / 788 / 245s
	B*-tree (no compaction)	1176ms	10% / 802 / 195s	10% / 796 / 236s	100% / 841 / 136s	100% / 866 / 135s
	B*-tree (with compaction)	1162ms	8% / 798 / 294s	18% / 821 / 284s	100% / 838 / 133s	100% / 867 / 133s

**Table 8: Comparison of sequence pair and B\*-tree on clustered floorplanning instances generated by floorplacement on IBM-MSwPins benchmarks. ibm05 does not have any macros.**

	# inst-ances	max / min # blocks	avg # blocks	% of successful instances	
				sequence pair	B*-tree
ibm01	42	49 / 1	18.9	35 / 83.3%	33 / 78.6%
ibm02	30	76 / 2	26.3	23 / 76.7%	21 / 70.0%
ibm03	39	49 / 1	22.9	28 / 71.8%	27 / 69.2%
ibm04	65	146 / 1	22.2	50 / 76.9%	46 / 70.8%
ibm06	49	46 / 1	19.8	37 / 75.5%	37 / 75.5%
ibm07	53	48 / 1	17.3	39 / 73.6%	40 / 75.5%
ibm08	71	133 / 1	16.5	59 / 83.1%	57 / 80.3%
ibm09	27	128 / 1	26.3	18 / 66.7%	18 / 66.7%
ibm10	117	68 / 1	26.2	93 / 79.5%	90 / 76.9%
ibm11	48	121 / 1	22.6	34 / 70.8%	32 / 66.7%
ibm12	121	113 / 1	19.1	95 / 78.5%	100 / 82.6%
ibm13	57	86 / 1	25.2	36 / 63.2%	33 / 57.9%
ibm14	239	119 / 1	19.9	183 / 76.6%	183 / 76.6%
ibm15	87	377 / 1	28.5	59 / 67.8%	61 / 70.1%
ibm16	82	114 / 1	27.5	59 / 72.0%	56 / 68.3%
ibm17	106	117 / 1	28.0	73 / 68.9%	78 / 73.6%
ibm18	80	46 / 1	23.4	67 / 83.0%	64 / 80.0%

instances appear once in a while — typically when a large block triggers floorplanning in a region with many small macros (Capo currently does not cluster macros, but this can be implemented if needed). Again, all results for sequence pair and B\*-tree are close.

In Table 9 we break down the 239 instances generated from ibm14 by size to study the performance of sequence pair and B\*-tree at different levels of physical hierarchy in floorplacement. One may suspect that sequence pair and B\*-tree respond differently to clustered instances of different sizes, but Table 9 shows little difference at every min-cut level. This reinforces our earlier observation that sequence pair and B\*-tree perform very similarly.

## 5. CONCLUSIONS

Floorplan representations have been the center of floorplanning research in the last decade, and their geometric properties have been extensively investigated. However, empirical evaluations have been restricted to a small set of benchmarks, and anecdotal evidence points to several versions of MCNC benchmarks that generate incompatible area and wirelength results. To this end, our work offers a comprehensive comparison of two well-researched floorplan representations in stand-alone block packing and in the context of min-cut floorplacement. This comparison is facilitated by several technical results presented in our work, such as the evaluation of floorplan slack in terms of B\*-tree and the simplification

**Table 9: Comparison of sequence pair and B\*-tree on clustered floorplanning instances generated by floorplacement on the ibm14 benchmark.**

min-cut level	# inst-ances	max / min # blocks	avg # blocks	# (%) of successful instances	
				sequence pair	B*-tree
5	2	119 / 51	85.0	1 / 50%	1 / 50%
6	3	88 / 59	70.7	2 / 66.7%	3 / 100%
7	5	47 / 32	39.4	3 / 60%	3 / 60%
8	21	49 / 3	31.0	18 / 85.7%	19 / 90.5%
9	47	49 / 1	26.9	38 / 80.9%	38 / 80.9%
10	69	49 / 1	19.0	57 / 82.6%	57 / 82.6%
11	72	36 / 1	10.6	48 / 66.7%	46 / 63.9%
12	16	31 / 1	10.8	12 / 75%	12 / 75%
13	3	1 / 1	1.0	3 / 100%	3 / 100%
14	1	1 / 1	1.0	1 / 100%	1 / 100%
total	239	119 / 1	19.9	183 / 76.6%	183 / 76.6%

of block-packing instances generated by min-cut floorplacement.

Our experiments strongly suggest that many theoretical results in block packing, traditionally formulated in the context of simulated annealing, have little relevance in digital design. For example, we show that a well-known algorithm for evaluating sequence pairs [21] runs in  $\Theta(n^{1.3})$  time in practice, which is better than its estimated worst-case complexity  $O(n^2)$ . Given that practical block-packing instances rarely exceed 300 blocks, this algorithm empirically outperforms a linear-time algorithm for B\*-tree, but the difference is dwarfed by wirelength evaluation. Other popular properties of representations, such as redundancies in the solution space, make little impact in the presence of wirelength optimization. Since congestion, delay and power usually take longer to compute than wirelength, interconnect evaluation in general is the bottleneck in annealing-based floorplanning. To this end, move-based incremental algorithms for interconnect evaluation can be very useful, but are difficult to design because even minor changes to relative block locations can significantly affect the structure of packed floorplans. Without incremental evaluation that is faster both asymptotically and empirically, new floorplan representations and fast evaluation algorithms seem irrelevant. Our conclusions are supported by extensive experiments on standard MCNC and GSRC benchmarks, as well as a rich set of block-packing instances derived from min-cut floorplacement of large mixed-size netlists.

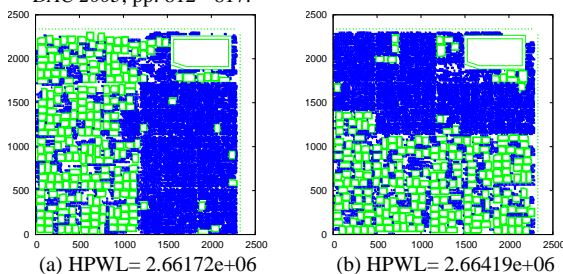
Algorithms that are not based on annealing may be competitive in area packing [4], but are often impractical because their objective function cannot be modified in applications. More generally, while we have not considered all possible algorithmic and design contexts for justifying new floorplan representations, changing the current *status quo* on their utility may require new breakthroughs.

**Table 7: Floorplacement using sequence pair and B\*-tree respectively on IBM-MSwPins mixed-size benchmarks. Capo is run on a 3.2GHz Linux workstation with 1GB RAM. Best HPWL for each benchmark is boldfaced. ibm05 does not have any macros.**

	sequence pair			B*-tree					
				with compaction			no compaction		
	HPWL (e6) / time (s) / %			time used in floorplanning / no. successful			FP instances / no. FP instances		
ibm01	2.67 / 155	35.9%	53/61	2.68 / 168	38.3%	44/51	<b>2.61 / 149</b>	32.2%	49/56
ibm02	5.52 / 458	50.2%	35/41	<b>5.39 / 567</b>	59.0%	30/36	5.40 / 285	14.1%	32/36
ibm03	8.78 / 464	44.5%	26/29	9.14 / 440	32.6%	44/51	<b>8.53 / 462</b>	30.6%	24/29
ibm04	<b>9.34 / 869</b>	52.6%	55/61	10.45 / 727	52.8%	45/53	10.11 / 715	53.4%	54/59
ibm06	<b>7.21 / 469</b>	30.0%	25/32	7.59 / 534	21.7%	27/29	7.38 / 501	16.6%	21/24
ibm07	<b>12.84 / 912</b>	46.8%	54/62	13.56 / 933	37.6%	34/45	14.05 / 1077	46.7%	16/22
ibm08	15.73 / 1498	40.7%	43/50	<b>15.45 / 2123</b>	70.2%	41/46	17.78 / 1854	55.2%	23/26
ibm09	16.46 / 1536	48.5%	15/20	16.63 / 1218	27.1%	35/44	<b>16.36 / 1117</b>	30.2%	28/35
ibm10	<b>34.36 / 1931</b>	34.0%	150/175	36.03 / 2279	49.6%	153/187	40.20 / 5430	40.8%	158/177
ibm11	22.69 / 1470	27.4%	74/86	22.49 / 1578	24.1%	66/78	<b>22.48 / 1462</b>	24.2%	64/73
ibm12	<b>41.86 / 1941</b>	23.9%	90/96	46.60 / 1983	34.0%	79/87	43.06 / 1894	28.8%	87/98
ibm13	<b>27.55 / 1857</b>	37.0%	50/56	29.13 / 2668	47.4%	46/52	27.93 / 2138	35.1%	74/89
ibm14	42.56 / 3085	22.3%	178/216	<b>41.62 / 2955</b>	19.7%	206/246	41.93 / 3355	29.0%	200/231
ibm15	<b>58.39 / 4353</b>	19.7%	89/100	58.78 / 4187	12.5%	68/83	58.86 / 4078	18.3%	100/112
ibm16	65.31 / 4912	17.0%	63/63	<b>65.21 / 4620</b>	21.5%	56/68	66.24 / 4447	16.6%	76/86
ibm17	<b>78.59 / 4138</b>	7.48%	94/101	78.67 / 3974	4.90%	105/116	78.73 / 4185	9.45%	113/121
ibm18	50.77 / 3886	7.78%	60/69	<b>49.97 / 3885</b>	9.58%	40/40	50.01 / 3584	4.10%	55/62
Average	0% / 0%			+2.24% / +5.57%			+2.40% / +7.61%		

## 6. REFERENCES

- [1] S.N. Adya and I.L. Markov, "Fixed-outline Floorplanning: Enabling Hierarchical Design," *IEEE Trans. on VLSI* 11(6), pp.1120-35, 2003. <http://vlsicad.eecs.umich.edu/BK/parquet/>
- [2] S.N. Adya, S.Chaturvedi, J.A. Roy, D.A. Papa and I. L. Markov, "Unification of Partitioning, Floorplanning and Placement," *ICCAD 2004*, pp. 550-557.
- [3] A. E. Caldwell, A. B. Kahng, I. L. Markov, "Optimal Partitioners and End-case Placers for Standard-cell Layout," *IEEE Trans. on CAD* 19(11), pp. 1304-1314, 2000.
- [4] H. H. Chan, I. L. Markov, "Practical Slicing and Non-slicing Block-Packing without Simulated Annealing," *ACM/IEEE Great Lakes Symp. on VLSI 2004*, pp. 282-287.
- [5] Y.-C. Chang, Y.-W. Chang, G.-M. Wu and S.-W. Wu, "B\*-trees: A New Representation for Non-Slicing Floorplans," *DAC 2000*, pp. 458-463.
- [6] K. Fujiyoshi and H. Murata, "Arbitrary Convex and Concave Rectilinear Block Packing Using Sequence Pair," *ISPD 1999*, pp. 103-110.
- [7] P.-N. Guo, C.-K. Cheng and T. Yoshimura, "An O-tree Representation of Non-Slicing Floorplan," *DAC '99*, pp. 268-273.
- [8] X. Hong et al., "Corner Block List: An Effective and Efficient Topological Representation of Non-Slicing Floorplan," *ICCAD 2000*, pp. 8-13.
- [9] A. B. Kahng, "Classical floorplanning harmful?" *ISPD 2000*, pp. 207-213.
- [10] R. E. Korf, "Optimal Rectangle Packing: New Results," *ICAPS 2004*, pp. 142-149.
- [11] M. Lai and D. Wong, "Slicing Tree Is a Complete Floorplan Representation," *DATE 2001*, pp. 228-232.
- [12] H.-C. Lee, Y.-W. Chang, J.-M. Hsu, and H. H. Yang, "Multilevel Floorplanning/Placement for Large-Scale Modules using B\*-trees," *DAC 2003*, pp. 812 - 817.
- [13] J.-M. Lin and Y.-W. Chang, "TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans," *DAC 2001*, pp. 764-769.
- [14] J.-M. Lin and Y.-W. Chang, "TCG-S: Orthogonal Coupling of P\*-admissible Representations for General Floorplans," *DAC 2002*, pp. 842-847.
- [15] P. H. Madden, "Reporting of Standard Cell Placement Results," *IEEE Trans. on CAD* 21(2), Feb. 2002, pp. 240-247.
- [16] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, "VLSI Module Placement Based on Rectangle-Packing by the Sequence Pair," *IEEE Trans. on CAD* 15(12), pp. 1518-1524, 1996.
- [17] H. Murata and E. S. Kuh, "Sequence-Pair Based Placement Methods for Hard/Soft/Pre-placed Modules," *ISPD 1998*, pp. 167-172.
- [18] Y. Pang, C.-K. Cheng and T. Yoshimura, "An Enhanced Perturbing Algorithm for Floorplan Design Using the O-tree Representation," *ISPD 2000*, pp. 168-173.
- [19] S. Prestwich, "Supersymmetric Modelling for Local Search," *SymCon '02*, September 2002. <http://user.it.uu.se/~pierref/astra/SymCon02/>
- [20] Z. C. Shen and C.C.N. Chu, "Bounds on the Number of Slicing, Mosaic, and General Floorplans," *IEEE Trans. on CAD* 22(10), pp. 1354 - 1361.
- [21] X. Tang, R. Tian and D. F. Wong, "Fast Evaluation of Sequence Pair in Block Placement by Longest Common Subsequence Computation," *DATE 2000*, pp. 106-111.
- [22] X. Tang and D. F. Wong, "FAST-SP: A Fast Algorithm for Block Placement Based on Sequence Pair," *ASPAC 2001*, pp. 521-526.
- [23] D. F. Wong and C. L. Liu, "A New Algorithm For Floorplan Design," *DAC 1986*, pp. 101-107.
- [24] G.-M. Wu and Y.-C. Chang and Y.-W. Chang, "Rectilinear block placement using B\*-trees," *ACM Trans. on Design Autom. of Electronic Systems* 8(2), pp. 188-202, 2003.
- [25] B. Yao et al., "Floorplan Representations: Complexity and Connections," *ACM Trans. on Design Autom. of Electronic Systems* 8(1), pp. 55-80, 2003.
- [26] E.F.Y. Young, C.C.N. Chu and Z.C. Shen, "Twin Binary Sequences: A Nonredundant Representation for General Nonslicing Floorplan," *IEEE Trans. on CAD* 22(4), pp. 457-469, 2003.
- [27] S. Zhou, S. Dong, C.-K. Cheng and J. Gu, "ECBL: An Extended Corner Block List with Solution Space including Optimum Placement," *ISPD 2001*, pp. 150-155.
- [28] H. Zhou and J. Wang, "ACG-Adjacent Constraint Graph for General Floorplans," *ICCD 2004*.
- [29] C. Zhuang, Y. Kajitani, K. Sakanushi and L. Jin, "An Enhanced Q-Sequence Augmented with Empty-Room-Insertion and Parenthesis Trees," *DATE 2002*, pp. 61-68.



**Figure 5: Sample placements produced by Capo 9.0 on ibm01 using (a) sequence pair and (b) B\*-tree. The discrepancies in wirelength versus Table 7 represent variability in Capo results.**