Fixed-outline Floorplanning : Enabling Hierarchical Design

Saurabh N. Adya, *Member, IEEE,* Igor L. Markov, *Member, IEEE*

***Abstract*— Classical floorplanning minimizes a linear combination of area and wirelength. When Simulated Annealing is used, e.g., with the Sequence Pair representation, the typical choice of moves is fairly straightforward.**

In this work, we study the fixed-outline floorplan formulation that is more relevant to hierarchical design style and is justified for very large ASICs and SoCs. We empirically show that instances of the fixed-outline floorplan problem are significantly harder than related instances of classical floorplan problems. We suggest new objective functions to drive simulated annealing and new types of moves that better guide local search in the new context. Wirelength improvements and optimization of aspect ratios of soft blocks are explicitly addressed by these techniques.

Our proposed moves are based on the notion of floorplan slack. The proposed slack computation can be implemented with all existing algorithms to evaluate sequence pairs, of which we use the simplest, yet semantically indistinguishable from the fastest reported [28]. A similar slack computation is possible with many other floorplan representations. In all cases the computation time approximately doubles.

Our empirical evaluation is based on a new floorplanner implementation Parquet-1 that can operate in both outline-free and fixed-outline modes. We use Parquet-1 to floorplan a design, with approximately 32000 cells, in 37 minutes using a top-down, hierarchical paradigm.

***Index Terms*— VLSI CAD, Physical Design, Floorplanning, Hierachical Design, Placement**

I. Introduction

We describe the classical floorplanning framework and compare it to a modern fixed-outline formulation.

A. Classical Outline-Free Floorplanning

A typical floorplanning formulation entails *a collection of blocks*, which can represent circuit partitions in applications. Each block is characterized by *area* (typically fixed) and *shape-type*, e.g., fixed rectangle, rectangle with varying aspect ratio, an L-shape, a T-shape, or a more general rectilinear polygon, etc (such shapes may optimize layouts of special types of circuits, e.g., datapaths). A solution to such a problem, i.e., a *floorplan*, specifies a selection of block shapes and overlap-free placements of blocks. Depending on shape constraints, a floorplanning formulation can be discrete or continuous. For example, if at least one block is allowed to assume any rectangular shape with fixed area and aspect ratio in the interval $[a,b]$ (where $a < b$) the solution space is no longer finite or discrete. Multiple aspect ratios can be implied by an IP block available in several shapes as well as by a hierarchical partitioning-driven design flow for ASICs [26], [13] where *only the number of standard cells* in a block (and thus the total area) is known in advance. In many cases, e.g., for row-based ASIC designs, there are only finitely many allowed aspect ratios, but solution spaces containing a continuum are used none the less, primarily because existing computational methods cannot handle such a large discrete solution space directly [13]. We point out that in the classical floorplanning formulations, movable blocks tend to have fixed aspect ratios, but the overall floorplan is not constrained by an outline. While several recent works allow for variable block aspect ratios, the more modern fixed-outline formulation (see below) has not been addressed.

Objective functions not directly related to area typically involve a *hypergraph* that connects given blocks. While more involved hypergraph-based objective functions have been proposed, the popularity of the HPWL (half-perimeter wirelength) function is due to its simplicity and relative accuracy, given that routes are not available. The HPWL objective became even *more relevant* [13] with the wide use of multi-layer over-the-cell routing in which more nets are routed with shortest paths.

A fundamental theorem for many floorplan representations, says that *at least one area-minimal placement can be represented* [20]. This does not hold for objectives that include wirelength because none of the optimal solutions may be "packed" which implies that more nets can be routed with shortest paths. Figure 1 shows a small example that area-minimal placements do not hold for minimum wirelength objectives. We also note that lack of incremental move structures in a floorplan representation is an important weakness of typical topological floorplanners. Thus wirelength has to be calculated from scratch after each floorplan evaluation. For designs with lot of wires, calculating the wirelength from scratch after each move can slow down the floorplanner considerably [1].

For the remaining part of this work, we will be dealing with the area and HPWL objectives only, but even this simplified setting implies *multi-objective op-*

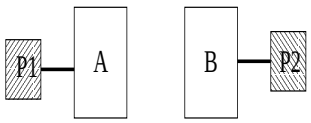
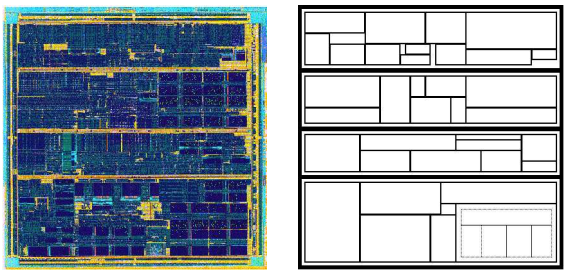


Fig. 1. **Example to show that area-minimal placements do not hold for minimum wirelength objectives. Blocks *A* and *B*, connected by 2-pin nets to fixed pins *P*1 and *P*2 respectively. The blocks touch in no optimal solution if the pins are sufficiently far from each other.**

timization. Mathematically, best trade-offs are captured by the *non-dominated frontier* (NDF), also known as the *Pareto front*.

Definition: a solution of a multi-objective optimization problem belongs to the *non-dominated frontier* iff no other solution improves upon one of the objective functions while preserving (or improving) other objective functions.[1] Of those works on abstract floorplanning that address both objectives, most minimize a linear combination [27], [23], [28] with arbitrarily chosen coefficients. By a simple corollary of the definition of NDF, this produces non-dominated solutions, most likely different for different coefficients. Note, however, that area and wirelength have different dimensions. Given that net lengths have the same order of magnitude as the x and y dimensions of the floorplan itself, areas tend to be several orders of magnitude larger than wirelengths and path delays. One can take the square root of the area so that the the terms become of the same magnitude. However, even if one tries to connect the area of a region to its perimeter, that relation depends on the aspect ratio (the rectangle 1x100 has perimeter length 202, and the rectangle 10x10 has perimeter length 40, even though both have area 100). Since aspect ratio changes in the course of floorplanning, one cannot come up with a fixed coefficient. Moreover, net length may be much larger than the perimeter because of the large number of nets. Here, again, there is no fixed coefficient because the net length changes during the course of floorplanning and it is difficult to predict optimal net lengths (some nets may be very short and some may be very long). The problem is exacerbated, because for different designs, different coefficients may be required to find the NDF. In our experiments, area terms dominated wirelength terms unless highly problem-specific coefficients are used. In other words, it is difficult to fully automate a floorplanner that explores non-dominated solutions with respect to wirelength and area objectives. The relationship between linear combination objectives and the Pareto curve (NDF) is studied in [12]. It is shown that with a suitable choice of coefficients, any point on the ”Lower Convex Hull” of the NDF can be found. This suggests a systematic method of modifying the coefficients to probe the hull and also characterizes the limitations of the linear combination approach.

To summarize, classical floorplan approaches entail difficult multi-objective optimization and often rely on representations that may not capture any minimum wirelength solutions.



(a) (b)

Fig. 2. **Layout of a modern chip. Figure (a) shows the actual layout. Figure (b) shows an abstract floorplan captured from the layout in (a). The example illustrates the hierarchical nature of designs and the need to support a hierarchical flow.**

B. Modern Fixed-outline Floorplanning

As pointed out in previous works [13], [4], some of fundamental difficulties in classical floorplanning are gracefully resolved in the context of modern ASIC design. Modern hierarchical ASIC design flows based on multi-layer over-the-cell routing naturally imply **fixed-die** placement and floorplanning rather than the **variable-die** style, associated with channel routing, two layers of metal and feedthroughs. Each top-down step of such a flow may start with a floorplan of prescribed aspect ratio, and with blocks of bounded (but not fixed) aspect ratios. The objective is to minimize wirelength subject to (i) the fixed floorplan outlines, and, perhaps (ii) zero white-space. Floorplans with no white-space are called “mosaic” by Hong et al. [11]. (i) implies that the white-space is no longer an objective, but rather a constraint, because it can be computed in advance. Modern design flows use hierarchy as a means to reduce the complexity. Figure 2 (a) shows the layout of a modern chip. Figure 2 (b) abstracts the actual layout in (a) to generate a hierarchical floorplan. This example serves to illustrate the need to support a hierarchical flow.

The modern floorplanning formulation was proposed by Kahng [13] and is an “inside-out” version of the classical outline-free floorplanning formulation — the aspect ratio of the floorplan is fixed, but the aspect ratios of the blocks can vary. To our knowledge, it has not yet been explicitly addressed in the literature, partly due to the lack of benchmarks. Since our work ad-

[1]The design of optimization heuristics can be viewed as a problem with at least two objective functions — runtime and solution quality.

dresses this formulation, we re-evaluate the relevance of classical floorplanning results in the new context:

1) Zero white-space requirement is practical because at top level (during floorplanning) there are no unused resources — the space between blocks can be used for routing, buffers, etc. For example, "buffer islands" are discussed in [7]. Therefore, whitespace must be allocated more carefully. The new formulation makes research on classical "block packing" *more relevant*. That is because all wirelength-minimal solutions in this formulation can be captured by compacted representations such as sequence pairs [20], *O*-trees [23], B^*-trees [5] and corner block lists [11]. In fact, *any* floorplan with zero white-space can be captured by known representations, because it is "compacted".
2) Multi-objective minimization of area and wirelength, via linear combinations or otherwise, is no longer an issue since white-space is fixed.
3) Handling blocks with variable aspect ratios appears increasingly important because there may be very few or no floorplans with a given outline for any given fixed configuration of aspect ratios. A number of works [19], [21], [6], [29] handle the floorplan sizing problem, i.e., changes of aspect ratios without reordering blocks, by methods of mathematical optimization (convex linear and non-linear programming). However, such methods are difficult to combine with combinatorial optimization and entail excessive runtimes. For example [29] cites runtime of 19.5 hours for the ami49 benchmark (other works cite smaller runtimes). Additionally, such approaches entail a mix of two very different computational engines. The implementation reported in [11] appears to handle discrete variable aspect ratios by randomized re-instantiation of blocks based on a set of 16 alternatives.
4) Perhaps, the greatest shortcoming of known approaches to floorplanning with respect to the new formulation is the lack of appropriate neighborhood structures, i.e., incremental changes ("moves") that preserve the fixed outline of the floorplan. Notably, every floorplan encoded by the corner block list(CBL) representation [11] has zero white-space with respect to the "rooms" the floorplan creates (i.e. is "mosaic"), but CBL based moves can change the floorplan's aspect ratio considerably.
5) Given that the new floorplanning formulation is *more constrained*, we see increased relevance of research on accommodating application-specific constraints, such as alignment, abutment, order, regions [28], symmetry [24], etc.

We conclude that classical floorplanning is largely relevant to the new floorplan formulation proposed by Kahng [13], however the new formulation must be addressed through ways other than novel representations. This is primarily due to the fact that *known floorplan representations and manipulation algorithms do not allow effective traversals of the solution space without violating important constraints*, such as the fixed-outline constraint discussed in our work. While such representations and algorithms may be proposed in the future, an alternative approach is to allow temporary violations and either tolerate or fix them. For example, not every corner block list [11] yields a valid floorplan, but the feasibility constraint is clearly stated in [11] and tolerated by the reported implementation. Constrained modern floorplanning has also been addressed recently by Feng et. al [9]. Their work assumes initial locations of blocks to be floorplanned are available and the techniques are more applicable for incremental floorplanning. There are a number of works that do floorplanning with various realistic objectives (congestion, timing, power, etc). However fixed-outline constraints and the optimization of the HPWL present a simpler, but necessary part of practical floorplanning. Thus, we view simplified floorplanning formulations as a useful filter for promising computational techniques.

Industrial design instances, can be broadly classified into ASICs, SoCs, and Microprocessor. ASIC chips frequently contain a handful (1-20) of large macros, a moderate number (100s) of large multi-row cells, and many small standard cells — up to several million and increasing. ASIC chips typically have whitespace ranging from 40% to 80%

SoC designs are similar to ASIC designs, but with many more large macros in the placement area. In extreme cases the bulk of the design is concentrated in standard pre-designed library cores, RAMs, etc, with only a small fraction of movable logic providing minor control functions.

Microprocessor designs are generally laid out hierarchically, and this approach often leads to many small partitions. Some of these partitions are small standard-cell placement instances with very few fixed cells, and a small number of movable cells (< 10000).

Particularly note that many current designs are hierarchical and are designed top-down [14], [16], [8]. As pointed out in [8] floorplanning is becoming increasingly important for prototyping hierarchical designs. The top level in any hierarchical design flow may use a variable die. Variable-die floorplanning is employed in [25] because only one level of hierarchy goes through their floorplanner. However, fixed-die floorplanning offers new possibilities. Fixed-outline floorplanning could be incorporated in a top-down hierarchical flow employing multi-level floorplanning as described in Section III-F. It can also be used in

an ASIC floor-placement flow [1] to place mixed-size ASIC designs in a fixed-die context. A methodology to place standard-cell designs with numerous macros by combining floorplanning and standard-cell techniques is proposed in [1]. The proposed design flow is as follows:

- An arbitrary black-box (no access to source code required) standard-cell placer generates an initial placement.
- To remove overlaps between macros, a physical clustering algorithm constructs a fixed-outline floorplanning instance.
- A fixed-outline floorplanner, generates valid locations of macros.
- With macros considered fixed, the black-box standard-cell placer is called again to place small cells.

This design flow provides a somewhat new "killer application" for the many floorplanning techniques developed in the last five years and fixed-outline floorplanning formulations are relevant in the fixed-die context which is very popular in ASIC design. Floorplanning is heavily used in design flows for complex hierarchical SoC design today. Given the dominance of fixed-outline design, our techniques are applicable to many chips that are being designed today.

In this work, we study neighborhood structures for the well-known sequence pair representation. Our proposed *slack-based moves* are more likely to reduce the floorplan span in a given direction (H or V) than random pair-wise swaps and block rotations used in most works based on sequence pairs. Wirelength minimization and handling aspect ratios of soft blocks are also more transparent with slack-based moves.

The remaining part of the paper is organized as follows. Section II discusses the background on floorplanning and the sequence pair representation. We introduce the concept of floorplan-slack in Section III. It also discusses better local-search in the annealing context and special moves aimed at HPWL minimization and handling soft-blocks using slacks. Fixed-outline floorplanning and applications to hierarchical floorplan design is also explained. Section IV presents empirical validation of our work, and future directions are discussed in Section V.

II. Background: The Sequence Pair Floorplan Representation

An overwhelming majority of floorplanners rely on the Simulated Annealing framework [26] but differ by internal floorplan representations.

The sequence pair representation for classical floorplans of N blocks has been proposed in [20]. Unlike most new graph-based representations, it consists of two permutations (orderings) of the N blocks. The two permutations capture geometric relations between each pair of 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

$$(<..,a,..,b,..>,<..,a,..,b,..>) \Rightarrow a \text{ is to left of } b \quad (1)$$

$$(<..,a,..,b,..>,<..,b,..,a,..>) \Rightarrow a \text{ is above } b \quad (2)$$

In other words, every two blocks constrain each other in either vertical or horizontal direction. The sequence pair representation is shift-invariant since it only encodes pairwise relative placements. Therefore, placements produced from sequence pairs must be aligned to given horizontal and vertical axes, e.g., $x = 0$ and $y = 0$. Multiple sequence pairs may encode the same block placement, e.g., for three identical square blocks, both $(<a,c,b>,<c,a,b>)$ and $(<a,c,b>,<c,b,a>)$ encode the placement with a straight on top of c, and b aligned with c on the right.

The original work on the sequence pair representation [20] proposed an algorithm to compute placements from a sequence pair by constructing the horizontal (H) and vertical (V) constraint graphs. The H and V graphs have $N+2$ vertices each — one for each of N block, plus "the source" and "the sink". For every pair of blocks a and b there is a directed edge $a \rightarrow b$ in the H graph if a is to the left of b according to the sequence pair (Formula 1). Similarly there is a directed edge $a \rightarrow b$ in the V graph if a is above b according to the sequence pair (Formula 2) — exactly one of the two cases must take place. Vertices that do not have outgoing edges are connected to the sink, and vertices that do not have incoming edges are connected to the source. Both graphs are considered vertex-weighted, where the weights in the H graph represent horizontal sizes of blocks, and the weights in the V graph represent vertical sizes of blocks. Sources and sinks have zero weights.

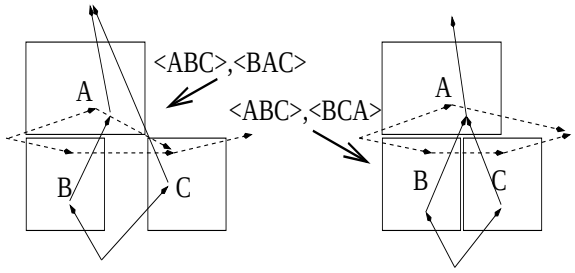


Fig. 3. **Two sequence pairs with edges of the horizontal (dashed) and vertical (solid) constraint graphs.**

Block locations are the locations of block's lower left corners. The x locations are computed from the H graph, and y locations are computed from the V graph *independently*. Therefore, we will only look at the computation of the x locations. One starts by assigning location $x = 0$ to the source. Then, the H graph is traversed in a topological order. To find the location

of a vertex, one iterates over all incoming edges and maximizes the sum of the source location and source width. Figure 3 illustrates the algorithm on two examples. The worst-case and average-case complexity of this algorithm is $O(n^2)$, since the two graphs, together, have a fixed $O(n^2)$ number of edges, and topological traversals take linear time in the number of edges.

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. A fundamental theorem from [20] implies that at least one minimal-area placement is representable with sequence pair (in fact, there are many). Therefore, sequence pair representation is justified for area minimization.

Sequence pairs can be used to floorplan hard rectangular blocks by Simulated Annealing [20], [21], [27], [28]. The moves are (i) random swaps of blocks in one of the two sequence pairs, and (ii) rotations of single blocks. Sequence pairs are modified in constant time, but need to be re-evaluated after each move. No incremental evaluation algorithms have been reported, therefore the annealer spends most of the time evaluating sequence pairs.

The sequence pair representation and necessary algorithms have been extended to handle fixed blocks [21] as well as arbitrary convex and concave rectilinear blocks [10]. Recently, the original $O(n^2)$-time evaluation algorithm [20], has been simplified and sped up to $O(n\log(n))$ by Tang et al. [27], and then to $O(n\log(\log(n)))$ [28]. Importantly, those algorithms do not change the semantics of evaluation — they only improved runtime, and lead to better solution quality by enabling a larger number of iterations during the same period of time. While *O*-trees [23] and corner block lists [11] can be evaluated in linear time, the difference in complexity is dwarfed by implementation variations and tuning, e.g., the annealing schedule. The implementation reported by Tang et al. [28] seems to outperform most known implementations, suggesting that the sequence pair is a competitive floorplan representation.

In our experiments, the simple $O(n^2)$ evaluation algorithm from [27] performs faster than the $O(n\log(n))$-time algorithm from the same paper. We implement the $O(n\log(n))$-time algorithm using C++ STL maps.[2] On ami49 benchmark with 49 blocks the $O(n\log(n))$-time algorithm is slower than $O(n^2)$-time algorithm by a factor of 7X. On a benchmark with 32498 blocks the $O(n\log(n))$-time is slower than $O(n^2)$-time algorithm by a factor of 4X. This is primarily due to the simplicity and lower implementation overhead of data structures used by the $O(n^2)$-time algorithm. A more recent paper [28] claims that their advanced $O(n\log(\log(n)))$-time algorithm outperforms the quadratic algorithm in practice. Given that it is considerably more involved, but based on the same principles, we choose to base our work on the quadratic algorithm, leaving out a potential speed up.

[2]STL maps are implemented using red-black trees.

All three algorithms are based on the following theorem [27]: The *x*-span of the floorplan to which sequence pair (S_1, S_2) evaluates, equals to the length of the longest common weighted subsequence of S_1 and S_2, where weights are copied from block widths. Analogous statement about the *y*-span deals with the longest common subsequence of S_1^R and S_2 , where ^{R} stands for "reversed" and weights are copied from block heights. Moreover, the computations of *x* and *y* locations of all blocks can be integrated into the longest common subsequence computations.

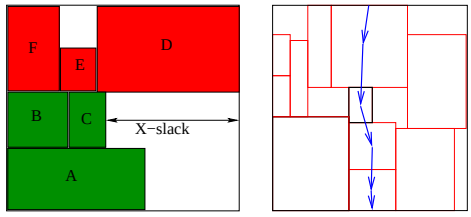


Fig. 4. **In Figure (a) X-slack of blocks B and C is shown by the solid arrow. Slack is the distance a block can be moved in a particular dimension without increasing the area of the floorplan. In Figure (b) blocks with zero Y-slack are shown. They lie on a "critical path" marked with arrows.**

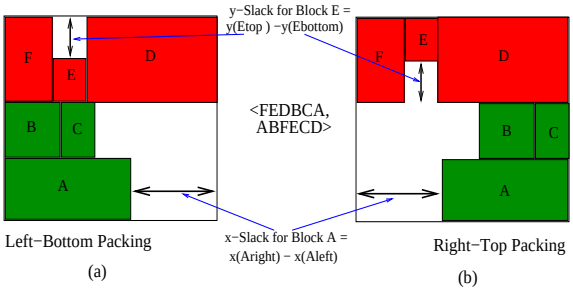


Fig. 5. **Slack Computation. In Figure (a) the floorplan is evaluated left-to-right and bottom-to-top. In Figure (b) the floorplan is evaluated right-to-left and top-to-bottom. The slacks for each block is the difference between its positions in the two evaluations.**

III. Better Local Search

We propose several ideas for improved move selection in Simulated Annealing and greedy floorplan optimization. We will also introduce our contribution in wirelength minimization and handling soft blocks.

A. Slack Computation

The notion of slack can be used with any of the above mentioned sequence pair evaluation algorithms

and potentially other floorplan representations. It is based on the following series of observations

- x and y locations are computed independently.
- in each dimension, the floorplan is constrained by one or more "critical paths" in respective constraint graphs. A critical path is a path of blocks that constrain each other in the same direction and are tightly packed so that any change in block location must produce overlaps or increase the span of the floorplan.
- in each dimension, the computation of block locations based on the constraint graphs is mathematically identical to the propagation of arrival times in Static Timing Analysis. Formally, STA is performed on an edge-weighted graph, while the constraint graphs are vertex-weighted, However, this difference is superficial since a vertex-weighted graph can be trivially transformed into an edge-weighted graph, e.g., by distributing vertex weights to incident edges, or otherwise.
- after the x-span X of the floorplan and x-locations of blocks are known, one can perform a symmetric computation of locations in right-to-left direction, assigning location X to the *sink* vertex. This will be analogous to the back-propagation of required arrival times in Static Timing Analysis.
- by analogy with Static Timing Analysis, the difference between the two locations computed for each block — slack — is related to the "most critical path" on which this block lies. In particular, zero slacks are always associated with paths that constrain the floorplan. Negative slacks are impossible as long as blocks do not overlap.

Slacks can be computed with any sequence pair evaluation algorithm that can work in left-to-right and right-to-left modes, which includes all algorithms we are aware of. Figure 5 shows floorplan evaluations for the same sequence pair in bottom-left mode and top-right mode.

A fast $O(n^2)$ algorithm for floorplan evaluation using sequence pairs was presented by Tang et. al [27]. It is based on Longest Common Subsequence (LCS) computation. The positions of blocks are recorded in left-to-right (bottom-to-top) order. The algorithm works as follows. Assume the blocks are $1...N$, and the input sequence pair is (X,Y). Both X and Y are a permutation of $1...N$. Block position array $Position[b], b = 1...N$, is used to record x or y coordinate of block b, depending on weight $w(b)$ equals to the width or height of block b respectively. To record the indices in both X and Y for each block b, the array $match[b], b = 1...N$ is constructed to be $match[b].x = i$ and $match[b].y = j$ if $b = X[i] = Y[j]$. The array $Length[1...N]$ is used to record the length of candidates of the LCS. The actual algorithm for LCS computation is shown in Figure 6. To calculate the actual positions of the blocks two calls to LCS_ORIG are made. X-positions are calculated by initializing the weights array $weights[1...N]$ with the widths of blocks and invoking LCS_ORIG($X,Y,xlocs$). Y-positions are calculated by initializing the weights array $w[1...N]$ with the heights of blocks and invoking LCS_ORIG($X^R,Y,ylocs$), where X^R is the sequence X in reversed order. Figure 7 shows the pseudo code to evaluate the floorplan in bottom-left mode, given a sequence pair (X,Y) using LCS computations.

To evaluate the floorplan in the top-right mode we need to evaluate the LCS of the two sequences in right-to-left order. This can be achieved easily by reversing the two sequences and invoking LCS_ORIG. The pseudo code for SP_EVAL_REV is presented in Figure 8. It reverses the two sequences, X and Y before calling the original algorithm. To compute the slacks we need the locations of the bottom-left corner of each block when evaluating the sequence pair in top-right mode. Lines 12 through 16 of SP_EVAL_REV achieve this.

Figure 9 presents pseudo code for the procedure **EVAL_SLACKS**, which evaluates slacks for each block in the design, given a sequence pair (X,Y). As illustrated in Figure 4 (a), the slack of a block in a floorplanning instance represents the distance (in a particular dimension) at which this block can be moved without changing the outline of the floorplan. Blocks with zero slack in the Y dimension are shown in Figure 4 (b). Such blocks must lie on critical paths in the relevant constraint graph. Figure 10 annotates blocks in a given floorplan with horizontal (x) and vertical (y) slacks.

The above mentioned code which evaluates slacks for blocks depends on the fact that LCS_ORIG(X,Y) and LCS_ORIG(X^R,Y^R) procedures give the same value for the length of the lcs of two sequences X and Y. We formalize this as a lemma below. The following notation is used; $lcs(X,Y)$ is the longest common sub-sequence of sequences X and Y and $LCS(X,Y)$ is the length of $lcs(X,Y)$

Lemma 1 *For two weighted sequences X and Y the length of lcs of X and Y is the same as the length of the lcs of reversed sequences X^R and Y^R.*

Proof The weights of individual blocks in the two sequence pairs (X,Y) and (X^R,Y^R) are the same. Thus by definition of lcs, $lcs(X,Y) = lcs(X^R,Y^R)$ and also $LCS(X,Y) = LCS(X^R,Y^R)$. Alternatively, as explained in [20], $LCS(X,Y)$ for a sequence pair (X,Y) representing a floorplan, is the length of the longest path in the horizontal constraint graph (HCG) implied by the sequence pair. i.e. the width of the floorplan. Reversing the two sequences to form the sequence pair (X^R,Y^R) amounts to reversing the direction of all the edges of the directed acyclic graph (DAG) represented by the HCG. Since the longest

```
1   LCS_ORIG(X,Y,Position,weights) /*Position[1...N] records block positions*/
2     for i = 1 to N   /*Initialize Match Array match*/
3     begin
4       match[X[i]].x = i;
5       match[Y[i]].y = i;
6     end
7     for i = 1 to N   /*Initialize Length Array Length with 0*/
8       Length[i] = 0;
9     for i = 1 to N
10    begin
11      b = X[i];
12      p = match[b].y;
13      Position[b] = Length[p];
14      t = Position[b] + weights(b);
15      for j = p to N
16        if(t > Length[j]) then Length[j] = t;
17        else break;
18    end
19  return Length[N];
```

Fig. 6. **Pseudo code for LCS_ORIG.** (X,Y) **is the input sequence pair. The computation is in left-to-right mode.**

```
1   SP_EVAL_ORIG(X,Y,xlocs,ylocs) /*xlocs,ylocs record block positions*/
2     for i = 1 to N  /*Initialize Weight Array*/
3       weights[i] = widths[i];
4     xSize = LCS_ORIG(X,Y,xlocs,weights); /*Save block x-locs in xlocs*/
5     for i = 1 to N  /*Initialize Weight Array*/
6       weights[i] = heights[i];
7     for i = 1 to N   /*Reverse X sequence*/
8       X^R[i] = X[N+1-i];
9     ySize = LCS_ORIG(X^R,Y,ylocs,weights);  /*Save block y-locs in ylocs*/
10  return;
```

Fig. 7. **Pseudo code for SP_EVAL_ORIG. It computes the location of blocks in bottom-left mode.** *xSize* **and** *ySize* **are the floorplan span.** $widths[1...N]$ **and** $heights[1...N]$ **hold the dimensions of blocks.**

```
1   SP_EVAL_REV(X,Y,xlocsRev,ylocsRev) /*xlocsRev,ylocsRev record block positions*/
2     for i = 1 to N  /*Reverse X sequence*/
3       X^R[i] = X[N+1-i];
4     for i = 1 to N  /*Reverse Y sequence*/
5       Y^R[i] = Y[N+1-i];
6     for i = 1 to N  /*Initialize Weight Array*/
7       weights[i] = widths[i];
8     xSize = LCS_ORIG(X^R,Y^R,xlocsRev,weights); /*Save block x-locs in xlocsRev*/
9     for i = 1 to N   /*Initialize Weight Array*/
10      weights[i] = heights[i];
11    ySize = LCS_ORIG(X,Y^R,ylocsRev,weights); /*Save block y-locs in ylocsRev*/
12    for i = 1 to N  /*Get bottom-left positions of blocks*/
13    begin
14      xlocsRev[i] = xSize - xlocsRev[i] - widths[i]
15      ylocsRev[i] = ySize - ylocsRev[i] - heights[i]
16    end
17  return;
```

Fig. 8. **Pseudo code for SP_EVAL_REV. It computes the location of blocks in top-right mode. SP_EVAL_REV reverses the two sequences of the sequence pair before calculating the LCS.**

```
1   EVAL_SLACKS (X,Y)  /*(X,Y) is the sequence pair*/
2     N = NumberOfBlocks;
3     xlocs[1...N], ylocs[1...N]  /*Block positions in bottom-left mode*/
4     xlocsRev[1...N], ylocsRev[1...N]  /*Block positions in top-right mode*/
5     xSlacks[1...N], ySlacks[1...N]  /*X and Y slacks for each block*/
6     SP_EVAL_ORIG(X,Y,xlocs,ylocs);
7     SP_EVAL_REV(X,Y,xlocsRev,ylocsRev);
8     for i = 1 to N
9     begin
10      xSlack[i] = xlocsRev[i] - xlocs[i];
11      ySlack[i] = ylocsRev[i] - ylocs[i];
12    end
13  return;
```

Fig. 9. **Pseudo code for EVAL_SLACKS. xlocs, ylocs, xlocsRev, ylocsRev hold the positions of blocks**

path in this new reversed HCG is the same as the old HCG, the length of lcs of (X^R, Y^R) should be the same as the length of the lcs of (X, Y). Similar argument holds for the vertical direction.

We now prove that the slacks for a block cannot be negative and blocks with zero slacks lie on the critical path.

Theorem 1

(a) *Slacks for a block cannot be negative.*

(b) *A block has zero slack in a particular dimension iff it lies on the "critical" path in that dimension.*

Proof

(a) We will base our discussion on the x-slack of a block. As computed by procedure EVAL_SLACKS (Figure 9), x-slack for each block is the difference in the x-locations of the block calculated in top-right mode and bottom-left mode. Let b be a block in the design. The sequence pair (X, Y) can be represented as $((X_l, b, X_r), (Y_l, b, Y_r))$. Similarly the reversed sequence pair (X^R, Y^R) can be represented as $((X_r^R, b, X_l^R), (Y_r^R, b, Y_l^R))$. The x-location of block b in the bottom-left mode is given by,

$xlocBL_b = LCS(X_l, Y_l)$ [28].

Similarly the x-location of block b in top-right mode is given by, $xlocTR_b = xSize - width_b - LCS(X_r^R, Y_r^R)$, where $xSize$ is the x-span of the floorplan. i.e. $xSize = LCS(X,Y) = LCS(X^R, Y^R)$. According to Lemma 1, $LCS(X_r^R, Y_r^R) = LCS(X_r, Y_r)$. Thus $xlocTR_b$ can be written as,

$xlocTR_b = LCS(X,Y) - width_b - LCS(X_r, Y_r)$

For two sequences $X = (X_l, b, X_r)$ and $Y = (Y_l, b, Y_r)$, if $LCS(X,Y) < LCS(X_l, Y_l) + width_b + LCS(X_r, Y_r)$, then the lcs of sequences X and Y is $(lcs(X_l, Y_l), b, lcs(X_r, Y_r))$, and the length of this lcs is $LCS(X_l, Y_l) + width_b + LCS(X_r, Y_r)$, which cannot be greater than $LCS(X,Y)$. Thus, by contradiction, the following inequality holds.

$LCS(X,Y) \geq LCS(X_l, Y_l) + width_b + LCS(X_r, Y_r)$

$\Rightarrow (LCS(X,Y) - width_b - LCS(X_r, Y_r)) - LCS(X_l, Y_l) \geq 0$

$\Rightarrow xlocTR_b - xlocBL_b \geq 0$

$\Rightarrow xSlack_b \geq 0$

We have proved that for any block b, its x-slack is non-negative. This holds true for the y-slack too. QED.

(b) As above, we base our discussion on the x-critical path. A *critical* path of a floorplan in x-dimension is defined as the longest x-path of the floorplan. There can be more than one critical path. However the length of all the critical paths is the same and equal to the $LCS(X,Y)$, where (X,Y) is the sequence pair representing the floorplan. If a block b lies on the x-critical path then b is a part of the lcs of X and Y. As shown in theorem 1(a), the following equality holds.

$LCS(X,Y) = LCS(X_l, Y_l) + width_b + LCS(X_r, Y_r)$

$\Rightarrow (LCS(X,Y) - width_b - LCS(X_r, Y_r)) - LCS(X_l, Y_l) = 0$

$\Rightarrow xlocTR_b - xlocBL_b = 0$

$\Rightarrow xSlack_b = 0$

Similarly if a block b has zero x-slack, then

$xlocTR_b - xlocBL_b = 0$

$\Rightarrow (LCS(X,Y) - width_b - LCS(X_r, Y_r)) - LCS(X_l, Y_l) = 0$

$\Rightarrow LCS(X,Y) = LCS(X_l, Y_l) + width_b + LCS(X_r, Y_r)$

$\Rightarrow b$ is on the longest x-path or the *critical* path.

The same argument holds for blocks on the y-critical path. QED.

Based on the above theorem we have the following corollary.

Corollary *If a move improves the floorplan span in x or y direction then it must involve some 0-slack blocks.*

Proof Let a move M improve the x-span of the floorplan represented by sequence pair (X,Y). Let (X_N, Y_N) be the sequence pair after the move. M could be of any type. e.g. swap, rotate etc. If the move M does not involve any block on the x-critical path, then $LCS(X_N, Y_N) \geq LCS(X,Y)$ because $lcs(X,Y)$ is also a sub-sequence of (X_N, Y_N). Thus, a move M must involve a block with zero x-slack in order to improve

the floorplan's x-span. Similar result holds for the y-direction. QED.

B. Slack-based moves

Once slacks are known, they can be used in move selection. Both the *timing analysis* interpretation above and the *common subsequence* interpretation from [27] imply that if a move (such as pair-wise swap) does not involve at least one block with zero slack in a given dimension, then the floorplan span in that dimension cannot decrease after the move. This is because such a move cannot improve critical paths or, equivalently, longest common subsequences. Therefore we bias move selection towards blocks having zero slack in at least one dimension. Of those blocks, the ones with large slack in the other direction are potentially good candidates for single-block moves, such as rotations, and more gradual aspect ratio changes, — discrete or continuous — can be chosen efficiently. Blocks with two zero slacks, especially small blocks, are good candidates for a new type of move, in which a block is moved simultaneously in both sequence pairs to become a neighbor of another block (in both sequence pairs, and, thus in placement). Namely, we attempt to move a critical block C next to a block L with as large slacks as possible, since large slacks imply that white-space can be created around L (more precise conditions can be written, but will still be heuristic). Figure 10 illustrates such a move. The following example illustrates the four possible ways of moving a block close to another by manipulating the sequence pair.

Example: Consider the five-block sequence pair $<< a,b,c,d,e > < c,a,d,e,b >>$. We wish to move block **e** close to block **a** in the floorplan. This can be done in four ways:

- $< \mathbf{a},\mathbf{e},b,c,d >, < c,\mathbf{a},\mathbf{e},d,b >$ (e is to right of a)
- $< \mathbf{e},\mathbf{a},b,c,d >, < c,\mathbf{e},\mathbf{a},d,b >$ (e is to left of a)
- $< \mathbf{a},\mathbf{e},b,c,d >, < c,\mathbf{e},\mathbf{a},d,b >$ (e is below a)
- $< \mathbf{e},\mathbf{a},b,c,d >, < c,\mathbf{a},\mathbf{e},d,b >$ (e is above a)

In addition to changing the sequence pair, our implementation changes block orientation and aspect ratio based on current slacks. We observe that [22] already suggested the analogy with static timing analysis in the context of FPGA placement. However, their algorithms are rather different and explicitly rely on H and V constraint graphs, while our proposed algorithms do not.

C. Handling Soft Blocks Using Slack Information

In a floorplanning instance, soft blocks have a fixed area but an aspect ratio which is variable between certain pre-determined limits. We added slack-based move types to change aspect ratios of soft blocks. During annealing, at regular intervals, a procedure called **PackSoftBlocks** is invoked to shape the soft blocks to improve the area of the total floorplan. **PackSoftBlocks** adopts a **greedy** approach. A block with low (preferably zero) slack in one dimension and high slack in the other dimension is chosen. The height and the width of such a block is changed within allowable limits so that its size in the dimension of smaller slack is reduced (to increase the slack). Such moves are greedily applied to all soft blocks in the design. Figure 11 gives the pseudo code for the greedy procedure **PackSoftBlocks**.

D. Wirelength Minimization

In classical floorplanning, the global objective is to minimize wirelength and total area of the design. This implies multi-objective minimization. Typically, most simulated annealing based floorplanners use a linear combination of area and wirelength as an objective for the annealer. In our floorplanner, we too use a linear combination of area and HPWL to evaluate annealer moves. Since area and wirelength have different dimensions they need to be normalized to give a reasonable linear combination. In our implementation, the area term is normalized by the total area of all blocks, and the wirelength term is normalized by the current wirelength of the floorplan at every move. The Δ term in the simulated annealing algorithm is calculated as follows.

$$\Delta_{area} = \frac{area_{new} - area_{old}}{totalArea_{blocks}}$$

$$\Delta_{HPWL} = \frac{HPWL_{new} - HPWL_{old}}{HPWL_{old}}$$

$$\Delta = (1 - weight_{HPWL}) * \Delta_{area} + weight_{HPWL} * \Delta_{HPWL}$$

$weight_{HPWL}$ is the linear weight attributed to the HPWL term and its value is between 0 and 1. During the annealing, all moves for which Δ is negative are accepted. All moves with a positive Δ are accepted if $random < exp(\frac{-\Delta * temp_{initial}}{temp_{current}})$, where $random$ is a random number between 0 and 1. Thus the probability of accepting a bad move decreases as $temp_{current}$ is reduced.

Additional moves are designed to improve the wirelength. For a given block a, we calculate, using analytical techniques, its "ideal" location that would minimize quadratic wirelength of its incident wires.[3] We determine the ideal location (x_a, y_a) of block a which minimizes the following function.

$$\sum_{N \in a} \sum_{v \in N} (x_v - x_a)^2 + (y_v - y_a)^2$$

[3] Analytical techniques are used because they are fast and easy to implement.

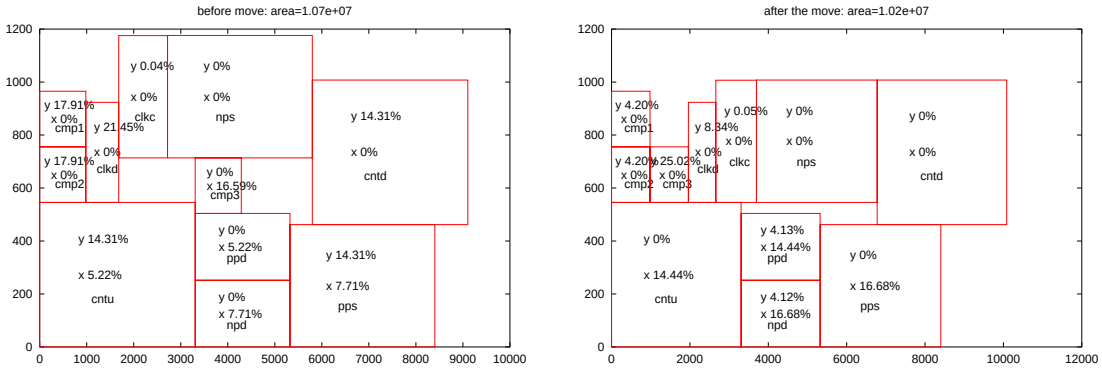


Fig. 10. **A slack-based move in a highly suboptimal floorplan of benchmark hp. x and y slacks are shown as percentages of the respective spans of the floorplan. Module `cmp3` — the smallest with zero y slack — is moved upon the module `cntu`, which has the high y slack. This move improves vertical span and slightly worsens the horizontal span, but the floorplan area is reduced.**

```
1   PackSoftBlocks (Direction)
2     Calculate X-Slacks and Y-Slacks for all N blocks (EVAL_SLACKS () );
3     Sort blocks in increasing order according to X-Slacks in sortedBlks_X-Slacks
4     Sort blocks in increasing order according to Y-Slacks in sortedBlks_Y-Slacks
5     for i = 1 to N
6     begin
7       if(Direction = Horizontal)
8         currBlock = sortedBlks_X-Slacks[i];
9       else if(Direction = Vertical)
10        currBlock = sortedBlks_Y-Slacks[i];
11      Shape currBlock to increase the slack in the critical direction &
        reduce the slack in the other direction, within allowable limits;
12    end
13  return;
```

Fig. 11. **Pseudo code for PackSoftBlocks. PackSoftBlocks is called once with Direction = Horizontal and once with Direction = Vertical.**

The ideal location (x_a, y_a) of block a is simply the average of the position of all modules connected to block a. We then identify the block b closest to the ideal location. This is done by expanding a circle centered at the ideal location and identifying the closest block b. We then attempt to move block a in the sequence pair so that in both sequences it is located next to b. As explained in Section III-B, we evaluate the four possible ways to do that, and choose the best. Thus an attempt is made to move a close to its ideal location to minimize quadratic wirelength.

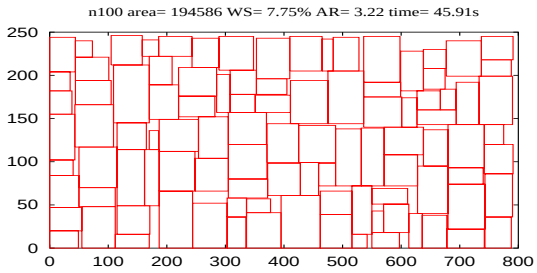


Fig. 12. **A floorplan with 100 blocks, generated without a constraining outline, has aspect ratio 3.22:1.**

Another type of move attempts to minimize both the floorplan size and wirelength objectives at the same time. Find a block b closest to the ideal location of the chosen block a such that the block b has large slack in at least one dimension. Depending on whether b has a large slack in the X-dimension or in the Y-dimension, we place a with a horizontal relation or a vertical constraint relative to b, respectively. Empirical measurements confirm that adding the proposed move types improves final floorplans.

E. Fixed-outline Constraints

Fixed-outline floorplans enable top-down design of very large scale ASICs and SoCs. Figure 12 shows the result of a floorplan with pure area minimization without any fixed outline constraints. The white-space achieved is 7.75% with an aspect ratio of 3.22:1. However this floorplan can be completely useless for a situation where 1:1 aspect ratio is imposed by a higher-level floorplan.

The following notation will be used in our floorplan-

ning formulations. For a given collection of blocks with total area A and given *maximum white-space fraction* γ, we construct a fixed outline with aspect ratio $\alpha \geq 1$.[4]

$$H_* = \sqrt{(1+\gamma)A\alpha} \qquad W_* = \sqrt{(1+\gamma)A/\alpha}$$

Aside from driving the annealer by area minimization, we consider the following objective functions:

1) The sum of the excessive length and width of the floorplan,
2) The greater of the excessive length and width of the floorplan.

Denoting the current height and width of the floorplan by H and W, we define these functions as

(1) $\max\{H-H_*,0\}+\max\{W-W_*,0\}$ **(2)** $\max\{H-H_*,W-W_*\}$

The choice of these functions is explained by the fact that the fixed-outline constraint is satisfied when and only when each of those functions takes value 0.0 or less. For this reason we cannot consider the product of fixed outline violations.

Our experiments show that a classic annealer-based floorplanner is practically unable to satisfy the fixed-outline constraints (for all of the three above-mentioned objective functions). Therefore we additionally bias the selection of moves. Figure 13 shows the evolution of the fixed-outline floorplan during Simulated Annealing with slack-based moves. The scheme works as follows. At regular time intervals during the simulated annealing the current aspect ratio is compared to the aspect ratio of the fixed outline. If the two are different, then the slack-based moves described earlier are applied to change the current aspect ratio in the needed direction. For example, if the width needs to be reduced then we chose the blocks in the floorplan with smallest slack in the x direction and insert them above or below the blocks with largest slack in the y direction. These moves have better chances of reducing the area and improving the aspect ratio of the current floorplan at the same time. Through these repeated moves during the simulated annealing the structure of the floorplan is biased towards the aspect ratio of the fixed outline. Our techniques also work with multi-objective minimizations (area+HPWL) and handle soft blocks in designs. As shown in Section IV, our implementation is successful in satisfying a variety of fixed-outline constraints. One concern about the intelligent move selection techniques during simulated annealing is that it may limit the solution space coverage. In general, these characteristics are necessary if simulated annealing is to be successful. We interleave the intelligent moves with totally randomized moves to ensure that simulated annealing does not get trapped in a local minima.

[4]The restriction of $\alpha \geq 1$ is imposed without loss of generality since our fborplanner can change orientations of individual blocks.

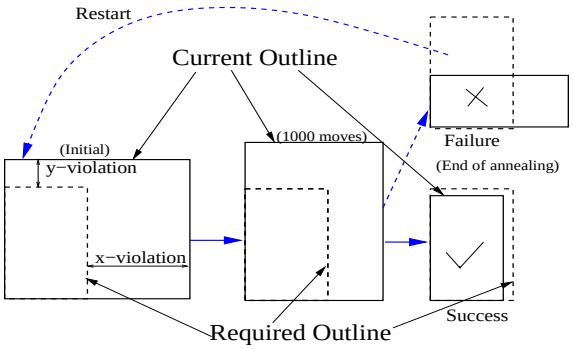


Fig. 13. **Snap-shots from fixed-outline floorplanning. The number of annealing moves is fixed, but if the evolving floorplan fits within the required fixed-outline, annealing is stopped earlier. If at the end of annealing the fixed-outline constraints are not satisfied, it is considered a failure and a fresh attempt is made.**

F. Hierarchical Layout

Hierarchical design is becoming increasingly attractive as a way to manage design complexity [18], [15]. It is argued in [18] that hierarchy is needed for humans, not for algorithms. The need for hierarchical design makes it imperative that the whole design flow support a hierarchical design methodology. We argue that fixed-outline floorplanning is an integral component of such a multi-level hierarchical flow. As discussed in [15], the top level floorplan might need to be fixed early on in the design cycle to avoid costly iterations later. Once the top-level floorplan is finalized, the high-level blocks and their shapes impose a fixed-outline constraint on the lower levels of design. A design team might decide to implement each high-level block flat, in which case there would be no need for a fixed-outline floorplanning tool. However, if the design team decides to implement high-level blocks in a hierarchical fashion as well then fixed-outline floorplanning becomes necessary. With increasing chip sizes, such a scenario is not unrealistic.

We use our techniques in fixed-outline floorplanning to develop a hierarchical floorplanning flow which is justified for very large ASICS and SoCs. We employ a simple connectivity based clustering scheme to create a top-level of hierarchy with clustered blocks. Each top-level clustered block is soft with aspect ratios allowed from 0.75 to 1.5 and has an area equal to $1.15 * sum\ of\ area\ of\ all\ sub-blocks$. Thus a white-space of 15% is allotted to each top-level block, so that Parquet can find a solution satisfying the fixed-outline constraints. The connectivity based clustering we use, is a multi-level greedy approach employing a series of passes till the design reduces to manageable size. In each pass we group highly connected cells together. Thus the design size reduces by a factor of 2 in each pass. We remove any net that connects cells only within

a cluster. More involved clustering schemes, like multi-way partitioning, could be employed for better HPWL minimization. We use these experiments to show that the fixed-outline floorplanner acts as an engine to enable hierarchical floorplanning. The top-level design is floorplanned without any fixed-outline constraints and the objective is to minimize area. The top-level clustered blocks impose fixed-outline constraints on the sub-blocks. Each top-level block is now floorplanned with these constraints.

Circuit	min/avg area (mm^2)	min/avg WS (%)	avg time (sec)
apte	46.60 / 47.73	0.09 / 2.42	8.75
xerox	19.39 / 20.06	0.24 / 3.52	7.75
hp	8.86 / 9.17	0.39 / 3.73	7.58
ami33	1.16 / 1.19	1.0 / 3.0	32.84
ami49	35.91 / 36.38	1.31 / 2.56	31

TABLE III

Outline-free, area minimization results with soft blocks. Slack-based moves are applied during annealing to modify the aspect ratio of soft blocks. All blocks have a variable aspect ratio. Averages and minima are over 100 independent starts.

Circuit	min/avg area (mm^2)	min/avg WS (%)	min/avg HPWL (mm)	avg time (sec)
apte	46.97/48.95	0.87/4.75	464/560	15.4
xerox	19.51/20.62	0.83/6.20	373/468	20.1
hp	8.96/9.72	1.50/8.96	177/214	15.3
ami33	1.18/1.24	2.43/7.05	62.5/75.4	31.0
ami49	36.07/37.8	1.75/6.20	673/812	31.9

TABLE IV

Outline-free, area + HPWL minimization results with soft blocks. Averages and minima are over 100 independent starts.

IV. Empirical Validation

We implement a floorplanner based on Simulated Annealing, **Parquet-1**. Runtimes are measured (in seconds) on a 1000MHz PC/Intel system that runs Linux. Implementations are in C++ and compiled with `g++ 2.95.2 -O3`.[5]

A. Annealing Schedule

Parquet-1 mostly follows a geometric cooling schedule. The initial temperature is chosen to be high enough (i.e. $temp_{initial} = 30000$) for most designs under consideration. For a design with N blocks, the temperature is decreased by a factor of $\omega < 1$ every $1.5N$ moves, as follows.

$$temp_{current} = \omega * temp_{old}$$

At certain deterministically defined temperatures, ω varies. The cooling is rapid in the initial phase (low value of ω) and very slow at the end (high value of ω). Thus most of the time during annealing is spent at low temperatures. ω is changed with temperature as follows.

$\omega = 0.85$ // $30000\ (temp_{initial}) > temp_{current} > 2000$
$= 0.90$ // $2000 > temp_{current} > 1000$
$= 0.95$ // $1000 > temp_{current} > 500$
$= 0.96$ // $500 > temp_{current} > 200$
$= 0.80$ // $200 > temp_{current} > 10$
$= 0.98$ // $10 > temp_{current} > 0.1\ (temp_{cool})$

There is also an option to run the annealer for a specified length of time. In this mode the temperature schedule remains the same but the number of moves between each iteration changes.

B. Classical Floorplanning Context

Table I compares **Parquet-1** to leading-edge floorplanning results on standard MCNC benchmarks in the area-only minimization context with no fixed-outline constraints. According to those results, our floorplanner is competitive with published implementations both in terms of final area and runtimes. We note, however, that all recently reported floorplanners easily achieve white-space well below 10%, therefore leaving very little possible improvement.

Table II shows results for simultaneous minimization of area and wirelength in a design. Results for different wirelength weights are presented.

Table III presents the area minimization results for designs with soft blocks. All blocks have a variable aspect ratio. Table IV presents the area and HPWL minimization results for designs with soft blocks.

In the following sub-section, we are going to show that fixed-outline floorplanning is significantly harder than outline-free floorplanning.

C. Fixed-outline floorplanning

The standard version of the floorplanner, without any of the slack based moves could not solve a single instance within the fixed outline, although it gave competitive area results. We tried different objective functions to drive the annealer as explained in Section III-E. When using the objective of minimizing the sum of excessive length and excessive width of the floorplan, the final aspect ratio of the floorplan is biased slightly towards the required aspect ratio. However, just changing the objective function was not powerful

[5]The C++ source code of Parquet is available on the Web at `http://vlsicad.eecs.umich.edu/BK/parquet/`

Circuit	Enh. O-Tree[23]		TCG[17]		CBL[11]		FastSP[28]		SP(**Parquet-1**)		
	area (mm^2)	time (sec)	area (mm^2)	time (sec)	area (mm^2)	time (sec)	area (mm^2)	time (sec)	min/avg area (mm^2)	min/avg white-space (%)	avg time (sec)
apte	46.92	11	46.92	1	NA	NA	46.92	1	47.07 / 48.14	1.08 / 3.28	4
xerox	20.21	38	19.83	18	20.96	30	19.80	14	19.83 / 20.73	2.42 / 6.65	3
hp	9.16	19	8.94	20	NA	NA	8.94	6	9.14 / 9.49	3.39 / 6.95	4
ami33	1.24	118	1.20	306	1.20	36	1.20	20	1.19 / 1.23	2.85 / 6.01	9
ami49	37.73	406	36.77	434	38.58	65	36.50	31	37.27 / 38.01	4.91 / 6.76	16

TABLE I

Outline-free area minimization results for Enhanced O-Tree(on Sun Ultra60), TCG(on Sun Ultra60, CBL(on Sun SPARC 20, Fast-SP(on Sun Ultra 1) and Parquet-1(on 1000MHz PC/Intel system). Averages and minima for Parquet-1 are over 100 independent starts.

Circuit	HPWL Weight = 0.1			HPWL Weight = 0.5			HPWL Weight = 0.9		
	min/avg area (mm^2)	min/avg HPWL (mm)	avg time (sec)	min/avg area (mm^2)	min/avg HPWL (mm)	avg time (sec)	min/avg area (mm^2)	min/avg HPWL (mm)	avg time (sec)
apte	47.08 / 49.75	530 / 591	10	47.31 / 50.50	516 / 563	10	55.87 / 65.82	478 / 495	10
xerox	19.98 / 20.99	376 / 544	10	20.07 / 21.61	372 / 466	10	20.23 / 22.97	360 / 397	10
hp	9.17 / 9.99	189 / 254	10	9.31 / 10.49	177 / 212	10	9.71 / 12.93	163 / 185	10
ami33	1.21 / 1.26	72.8 / 89.55	25	1.20 / 1.30	63.2 / 74.2	25	1.38 / 1.58	51.9 / 56.4	16
ami49	37.41 / 39.21	770 / 1026	29	38.67 / 40.51	727 / 841	29	40.6 / 45.49	589 / 659	30

TABLE II

Outline-free, area + HPWL minimization results. A linear combination of area and half perimeter wirelength is minimized during simulated annealing. Results for different HPWL weights are presented. Averages and minima are over 100 independent starts.

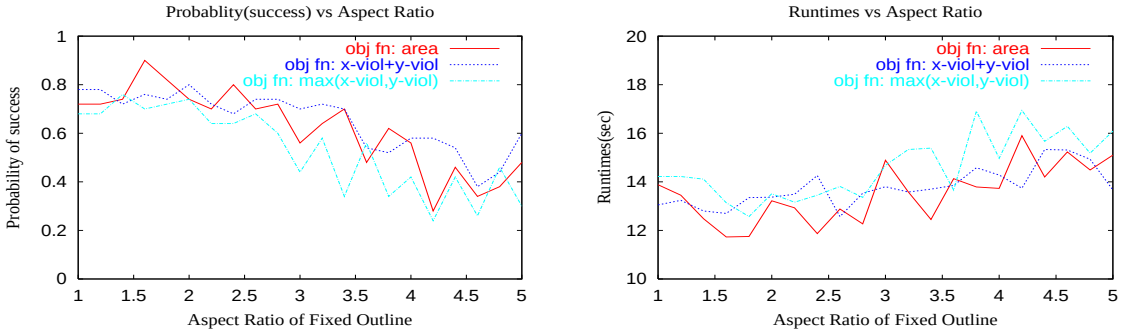


Fig. 14. **Probability of success and average runtimes for floorplanning design n100 with fixed-outline constraints performed by annealing with three alternative objective functions and slack-based moves. The maximum white-space for the design is 15%. i.e. $\gamma = 15\%$. In order to remove noise we plotted average of 50 runs for each aspect ratio.**

enough and the floorplanner could not satisfy the fixed-outline constraints for a single instance. This confirms the inadequacy of the classical min-area floorplanning formulation and algorithms in the fixed-outline context.

To achieve fixed-outline floorplan, we consider three objectives in terms of excessive height and width as described in Section III-E (the sum of and the greater of) and the area. We stop the annealer as soon as it finds a solution satisfying a given fixed outline. If the current outline is smaller, its aspect ratio can be different from the aspect ratio of the fixed outline. If the annealer's temperature schedule runs out and no satisfying solution is found, we deem this a failure.

We constrained our final solutions to have a maximum white-space of 15% and tried to achieve floorplans satisfying different fixed-outlines. Experiments are performed on **n100** benchmark and the results are averaged for 50 runs for each aspect ratio. Figure 14 shows plots of (i) *the probability of success* of satisfying the fixed outline constraint vs desired aspect ratio of the fixed outline, and (ii) *the average runtimes for all runs* vs the desired aspect ratio of fixed outline. The

plots reflect the difficulty in satisfying fixed-outline floorplans with given aspect ratios, which highly depends on the dimensions of the blocks. As seen from the plots, our simulated annealer fairly often failed to satisfy the given outline, however, the probability of success is typically over 50%, i.e., at least five in ten starts are successful. This consistent rate of success suggests that our slack-based moves indeed improve local search (simulated annealing without slack-based moves is never able to satisfy the fixed outline). Also note that in most of the unsuccessful attempts the final solutions are within 1-2% from the desired outline, yet *we regard them as failures*.

Out of the three objective functions we tried, minimizing the sum of excessive width and height and minimizing the area is more successful than minimizing the maximum of excessive width or height. Finding an explanation of this empirical result remains an open problem.

When we decreased γ in our experiments, some fixed outlines are never satisfied, which may be due to the absence of solutions with a given aspect ratio and very small white-space. The plot of probability of success of satisfying the fixed-outline constraint for a design(n100) with 12% white-space, is shown in Figure 15. As expected, decreasing γ worsens the probability of success. The average runtimes required to satisfy the fixed-outline constraints also increase with lower γ.

Figure 16 shows the plots of probabilities of success, HPWL and average time vs. aspect ratios for design n100 fixed-outline constraints in the wirelength minimization mode. The time taken to satisfy the fixed-outline constraints increases significantly compared to those in Figure14, because of the overhead of calculating the wirelength from scratch for every move. The probabilities of success decrease a little. However for very skewed aspect ratios, wirelength minimization suffers. In the **outline-free mode**, Parquet achieves an average wirelength of 323 and average white-space of 10.18% over 50 independent starts for the design n100. Our experiments with other publicly available benchmarks (n50, n200, ami49 etc) produced consistent results.

D. Hierarchical Layout Context

We develop a hierarchical floorplanning flow employing the fixed-outline floorplanner, Parquet, as an engine. For our experiments we use the publicly available[6] ibm06 placement benchmark [1]. ibm06 is a mixed-size placement benchmark with **32498** cells, including **178** macros. The standard cells are of varying widths but similar height and a modified standard cell placement algorithm which can handle macros should be employed to place such a design. However, we perform this experiment to demonstrate the scalability of hierarchical floorplanning. We consider all blocks to be hard, thus further constraining the problem. We employ a simple connectivity based clustering scheme to create a top-level of hierarchy with 238 clustered blocks (each top-level block having approximately 128 blocks). Big macros are kept out of this clustering. A white-space of 15% is allotted to each top-level clustered block. Each top-level clustered block is soft with aspect ratios allowed from 0.75 to 1.5. The top-level design is floorplanned without any fixed-outline constraints and the objective is to minimize area. The top-level floorplan is shown in Figure 17 (a). The top-level clustered blocks impose fixed-outline constraints on the sub-blocks. Each top-level block is now floorplanned with these constraints. The final flat floorplan of ibm06 is shown in Figure 17 (b). We achieved a dead-space of 17.44% in 37 minutes. In comparison floorplanning ibm06 design flat achieved a deadspace of 55.62% in 1070 minutes. We tried speeding up the flat floorplanning by using the $O(n\log(n))$ fast sequence pair evaluation algorithm [27]. However, as pointed out in Section II, $O(n\log(n))$ algorithm performed worse than $O(n^2)$ algorithm. While, in our experiments we only used a single level of hierarchy, the flow could be easily changed to handle multiple levels of floorplanning hierarchy. Also, we considered only area as an objective, but a linear combination of area and wirelength can also be considered as an objective. We perform the hierarchical floorplanning experiment to demonstrate that multi-level floorplanning is much more scalable than flat floorplanning. Indeed, floorplanning almost 32K objects flat using the sequence pair representation would require inordinate amounts of time. The floorplan obtained by hierarchical floorplanning can further be compacted using efficient layout compaction schemes or employing low temperature annealing. However, with close to 32K objects as in our case, it might be too costly to even try low temperature annealing with the sequence pair representation. We conclude that in comparison to the flat floorplanner, the hierarchical floorplanner scales much better in terms of runtime and solution quality. Thus, hierarchical floorplanning can used to floorplan a large number of top-level blocks efficiently and also to support a multi-level hierarchical design flow.

[6]The benchmarks are available on the Web at `http://vlsicad.eecs.umich.edu/BK/ISPD02bench/`

V. Conclusions

Our work makes an important step in identifying and evaluating fundamental optimization techniques that are successful for wirelength optimization in large-scale fixed-outline floorplanning. In perspective, we expect our techniques to be useful (i) in floorplanning with more realistic objective functions, (ii) during

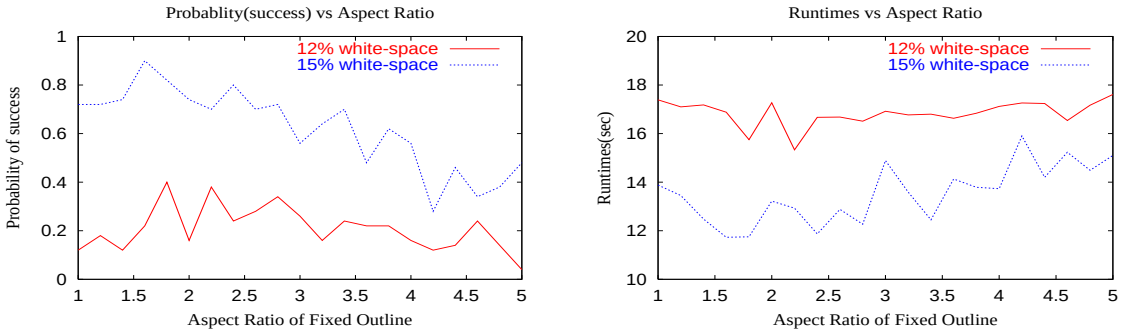


Fig. 15. **Probability of success and average runtimes for floorplanning design n100 with fixed-outline constraints. The objective function is area. The maximum white-space for the design is 12%. i.e.** $\gamma = 12\%$**. Probabilities and runtimes for a design with maximum white-space of 15% are also provided for comparison. The probabilities of success with 12% white-space are significantly lower compared to those with 15% white-space. because of smaller** γ**. We plotted average of 50 runs for each aspect ratio.**

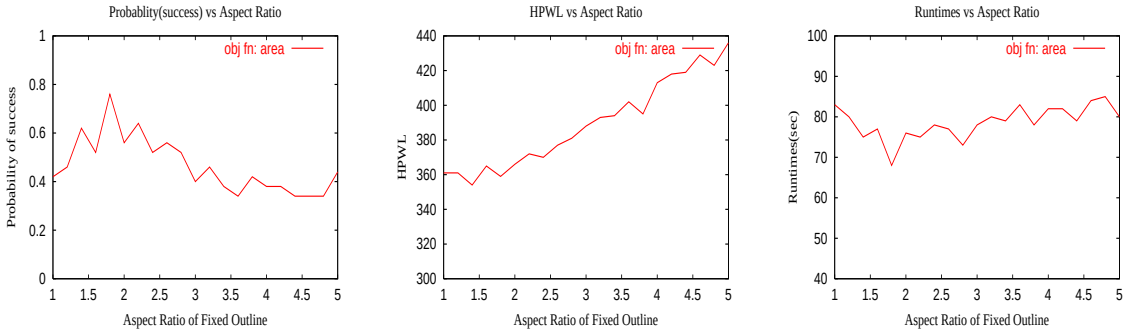
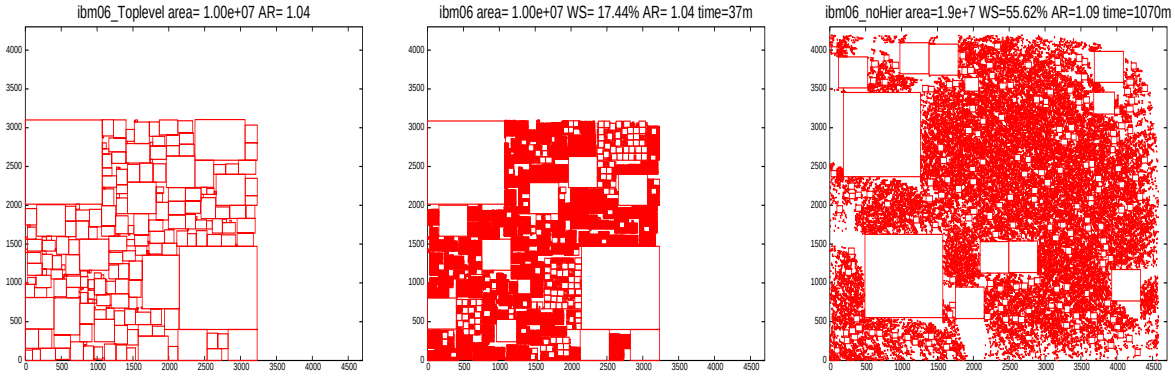


Fig. 16. **Probability of success, HPWL and average runtimes for floorplanning design n100 with fixed-outline constraints in HPWL minimization mode. The maximum white-space for the design is 15%. i.e.** $\gamma = 15\%$**. The probabilities of success are slightly lower compared to Figure 14, because of multi-objective minimization. Also, the wirelength minimization suffers when required aspect ratios are skewed. We plotted average of 50 runs for each aspect ratio.**



(a) (b) (c)

Fig. 17. **Hierarchical Floorplanning of ibm06 design with 32498 blocks. ibm06 is a mixed-size design and has 178 macros. We use a connectivity based clustering scheme to reduce the design size at top level to 238 clustered blocks (each top-level block having approximately 128 blocks). Big macros are kept out of this clustering. All blocks are hard. The top level is floorplanned without any fixed-outline constraints. The top-level blocks impose fixed-outline constraints on the lower level. The top-level floorplan is shown in Figure (a). The final flat floorplan is shown in Figure (b). 17% deadspace was achieved in 37m. In comparison Figure (c) shows the floorplan of ibm06 obtained by flat floorplanning. 55% deadspace was achieved in 1070m.**

related optimizations with additional degrees of freedom such as added logic [re-]synthesis, and (iii) in non-classical design flows such as virtual prototyping. These extensions will be explored in our future work.

This work points out that a non-standard floorplanning formulation — fixed-outline floorplanning is significantly harder than classic min-area outline-free floorplanning. We implement an annealing-based floorplanner **Parquet-1** that uses a recently discovered [27] sequence pair evaluation algorithm and study its performance both in the fixed-outline and outline-free contexts. We use the concept of *slacks* in a floorplan for better local search. We added special techniques, based on slacks of individual blocks in a floorplan, to handle soft blocks. These new techniques when incorporated into the simulated annealing framework perform well and achieve an area utilization close to 99% for MCNC benchmarks within reasonable time. We also introduce special moves based on analytical methods to better drive wirelength (HPWL) minimization during simulated annealing. For the standard formulation, our floorplanner is competitive, both in terms of runtime and solution quality, with other leading-edge implementations and represents current state-of-the-art. However, our implementation experiences serious difficulties in the fixed-outline context until the algorithm is modified. In particular, more relative white-space is required to satisfy an outline of a given area when its aspect ratio is fixed.

We propose new objectives that more successfully drive our annealing-based floorplanner to satisfy fixed-outline constraints. New types of slack-based moves, that may be applicable to most floorplanner implementations based on simulated annealing, are introduced. These special moves performed during annealing provide better control of the x and y dimensions of the floorplan. We study the sensitivity of relative white-space in the design on the effectiveness of our proposed methods. We also study the effect on wirelength minimization when trying to achieve various fixed-outlines.

Our experiments show that classical methods fail for fixed-outline instances constructed from standard MCNC benchmarks and other publicly available benchmarks, but when new objectives and slack-based moves are added to our **Parquet-1** implementation, it finds acceptable fixed-outline floorplans for a variety of aspect ratios. We also conclude that minimizing the sum of excessive width and height is a more successful approach than minimizing the greater of the two.

We demonstrate a top-down, hierarchical floorplanning flow with a single level of hierarchy. We are able to floorplan **32498** blocks and achieve a dead-space of 17.44% in 37 minutes. In comparison flat floorplanning achieved a deadspace of 55.62% in 1070 minutes.

We do not necessarily advocate our particular way of doing hierarchical floorplanning, and plan to study this our future work. We also note that large designs can be first partitioned using recursive bisection, but not necessarily all the way down to the level of detail where the differences in block sizes and shapes complicate recursive bisection. Our on-going work [1] aims to combine recursive bisection and floorplanning techniques in the context of mixed-mode placement. In our on-going research we are extending the proposed methods to top-down, multi-level hierarchical floorplanning and related applications to standard-cell placement with large macro cells.

REFERENCES

[1] S. N. Adya and I. L. Markov, "Consistent Placement of Macro-blocks Using Floorplanning and Standard-Cell Placement", *ISPD 2002*, pp. 12-17

[2] S. N. Adya, I. L. Markov, "Fixed-outline Floorplanning Through Better Local Search", *ICCD 2001*, pp. 328-334.

[3] A. E. Caldwell, A. B. Kahng, A. A. Kennings and I. L. Markov, "Hypergraph Partitioning for VLSI CAD: Methodology for Reporting, and New Results", *DAC 1999*, pp. 349-354.

[4] A. E. Caldwell, A. B. Kahng and I. Markov, "Can Recursive Bisection Alone Produce Routable Placements?", *DAC 2000*, pp. 477-482.

[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] P. Chen and E. S. Kuh, "Floorplan Sizing by Linear Programming Approximation", *DAC 2000*, pp. 468-471.

[7] J. Cong, T. Kong and D.Z. Pan "Buffer Block Planning for Interconnect-Driven Floorplanning," *ICCAD 1999*, pp. 358-363.

[8] W. Dai, D. Huang, C. Chang, M. Courtoy, "Silicon Virtual Prototyping: The New Cockpit for Nanometer Cip Design" *ASP-DAC 2003*, pp. 635-639.

[9] Y. Feng, D. P. Mehta and H. Yang "Constrained "Modern" Floorplanning",*ISPD 2003*, pp. 128-134

[10] K. Fujuyoshi and H. Murata, "Arbitrary Convex and Concave Rectilinear Block Packing Using Sequence Pair", *ISPD 1999*, pp. 103-110.

[11] X. Hong et al., "Corner Block List: An Effective and Efficient Topological Representation of Non-Slicing Floorplan", *ICCAD 2000*, pp. 8-13.

[12] A. Jagannathan, S. W. Hur and J. Lillis, "A Fast Algorithm For Context-Aware Buffer Insertion", *ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 7, No. 1,January 2002*, pp. 173-188

[13] A. B. Kahng, "Classical Floorplanning Harmful?", *ISPD 2000*, pp. 207-213.

[14] J. Koehl, D. E. Lackey, G. Doerre, "IBM's 50 Million Gate ASICs", *ASP-DAC 2003*, pp. 628-634.

[15] D. E. Lackey, " Design Planning Methodology for Rapid Chip Deployment", *IEEE/DATC Electronic Design Process Workshop, 2001*

[16] A. Mehrotra, L. V. Ginneken and Y. Trivedi, "Design Flow and Methodology for 50M gate ASIC" *ASP-DAC 2003*, pp. 640-647.

[17] J. Lin and Y. Chang, "TCG: A Transitive Closure Graph Based Representation for Non-Slicing Floorplans", *DAC 2001*, pp. 764-769.

[18] L. Scheffer, "Data Modeling and Convergence Methodology in Integration Ensemble", *IEEE/DATC Electronic Design Process Workshop, 2001*

[19] T. S. Moh, T. S. Chang and S. L. Hakimi, "Globally Optimal Floorplanning for a Layout Problem", *IEEE Trans. on Circuits and Systems - I*, vol 43, no. 9, pp. 713-720, September 1996.

[20] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence pair", *IEEE Trans. on CAD, 1996*, vol 15(12), pp. 1518-1524

[21] H. Murata and E. S. Kuh, "Sequence-Pair Based Placement Methods for Hard/Soft/Pre-placed Modules", *ISPD 1998*, pp. 167-172.
[22] S. Nag and K. Chaudhary, "Post-Placement Residual-Overlap Removal with Minimal Movement", *DATE '99*, pp. 581-586,
[23] 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.
[24] Y. Pang, F. Balasa, K. Lampaert and C. K. Chang, "Block Placement with Symmetry Constraint Based on the O-Tree Non-Slicing Representation", *DAC 2000*, pp. 464-468.
[25] F. Remond,"Monterey Usage at STMicroelectronics", *DAC 02*, `http://www.montereydesign.com/customers/ST%20Presentation%20DAC%202002.pdf`
[26] N. Sherwani, "Algorithms For VLSI Design Automation", *Kluwer*, 3rd ed. 1999.
[27] X. Tang, R. Tian and and D. F. Wong, "Fast Evaluation of Sequence Pair in Block Placement by Longest Common Subsequence Computation", *DATE 2000*, pp. 106-111.
[28] X. Tang and D. F. Wong, "FAST-SP: A Fast Algorithm for Block Placement Based on Sequence Pair", *ASPDAC 2001*.
[29] F. Y. Young, C. C. N. Chu, W. S. Luk and Y. C. Wong, "Floorplan Area Minimization Using Lagrangian Relaxation", *ISPD 2000*, pp. 174-179.

Saurabh N. Adya received his bachelor's degree in Electronics and Communication Engineering from KREC, Mangalore University, India in 1999 and a master's desgree in Computer Science and Engineering from the University of Michigan, Ann Arbor, in 2002. He is currently a doctoral candidate in the EECS department at the University of Michigan. From 1999 to 2000, he worked as IC Design Engineer at Texas Instruments, India. His current research interests are in the general area of VLSI CAD and specifically, physical design for VLSI.

Igor L. Markov is an assistant professor of Electrical Engineering and Computer Science at the University of Michigan. He did his master's (Math) and doctorate (CS) work at UCLA, where he earned the Best Ph.D. student award for 2000 at the Computer Science Department.

Prof. Markov's interests are in quantum computing and in combinatorial optimization with applications to the design and verification of integrated circuits. Prof. Markov's contributions include the circuit placer Capo and the quantum circuit simulator QuIDDPro.

Prof. Markov co-authored more than 50 publications and is serving on technical program committees at ICCAD, DATE, ISPD, GLSVLSI, SLIP and IWLS in 2003.