

Partitioning-based Methods for VLSI Placement

Jarrold A. Roy and Igor L. Markov

University of Michigan, EECS Department, Ann Arbor, MI 48109-2121

1 Introduction

The technique of using balanced min-cut partitioning in placement was presented by Breuer in 1977 [7]. Such min-cut placers use scalable and extensible divide-and-conquer algorithmic framework and tend to produce routable placements [9]. Recent work offers extensions to block placement and large-scale mixed-size placement [15, 18, 31], and robust incremental placement [33].

Over the years partitioning-based placement has seen many revisions and enhancements, but the underlying framework (illustrated in Figure 1) remains much the same. Top-down partitioning-based placement algorithms seek to decompose a given placement instance into smaller instances by sub-dividing the placement region, assigning modules to subregions and cutting the netlist hypergraph [7, 19]. The top-down placement process can be viewed as a sequence of passes where each pass examines all bins and divides some of them into smaller bins. Most commonly the division step is accomplished with balanced min-cut partitioning that minimizes the number of signal nets connecting modules in multiple regions [7]. These techniques leverage well-understood and scalable algorithms for hypergraph partitioning and typically lead to routable placements [9].

This chapter is organized as follows: Section 2 introduces the basic framework for min-cut partitioning-based placement. Next, Section 3 presents recent enhancements to min-cut placement. Section 4 describes adapting partitioning based methods to mixed-size placement and Section 5 outlines recent techniques that give min-cut placement an edge over other placement algorithms. Lastly, Section 6 discusses the workings of state-of-the-art min-cut placers such as Dragon [38, 39, 43], FengShui [4, 27], NTUPlace2 [22] and Capo [30–35] and illustrates how they differ from the basic min-cut framework and each other.

```

Variables: queue of placement bins
Initialize queue with top-level placement bin
1 While (queue not empty)
2   Dequeue a bin
3   If (bin small enough)
4     Process bin with end-case placer
5   Else
6     Choose a cut-line for the bin (including direction)
7     Build partitioning hypergraph from netlist
      and cells contained in the bin
8     Partition the bin into smaller bins
      (generally via min-cut bi- or quadri-section)
9     Enqueue each child bin

```

Figure 1: Top-down partitioning-based placement.

2 Top-down Partitioning-based Placement Framework

Using min-cut partitioning in placement was presented by Breuer in 1977 [7]. The underlying framework remains mostly the same since then and is illustrated in Figure 1. The placement region is represented by a series of *placement bins* which represent (i) a placement region with allowed module locations (*sites*), (ii) a collection of circuit modules to be placed in this region, (iii) all signal nets incident to the modules in the region, and (iv) fixed cells and pins outside the region that are adjacent to modules in the region (*terminals*).

Min-cut partitioning-based placers generally proceed by dividing the netlist and placement area into successively smaller pieces until the pieces are small enough to be handled efficiently by optimal end-case placers [11]. State-of-the-art placers generally use a wide range of hypergraph partitioning techniques to best fit partitioning problem size — optimal (branch-and-bound [11]), middle-range (Fiduccia-Mattheyses [20]) and large-scale (multi-level Fiduccia-Mattheyses [10, 26]). Min-cut placement is highly scalable (due in large part to algorithmic advances in min-cut partitioning [10, 20, 26]) and typically produces routable placements.

In this section, we introduce topics relevant to top-down partitioning-based placement that must be addressed by all modern min-cut placers. Specifically we discuss terminal propagation, bipartitioning vs. multi-way partitioning, cut-line selection and whitespace (or free space) allocation.

2.1 Terminal Propagation and Inessential Nets

Proper handling of terminals is essential to the success of top-down placement approaches [11, 19, 21, 37]. When a particular placement bin is split into multiple subregions, some of the cells inside may be tightly connected to cells outside of the bin. Ignoring such connections can adversely

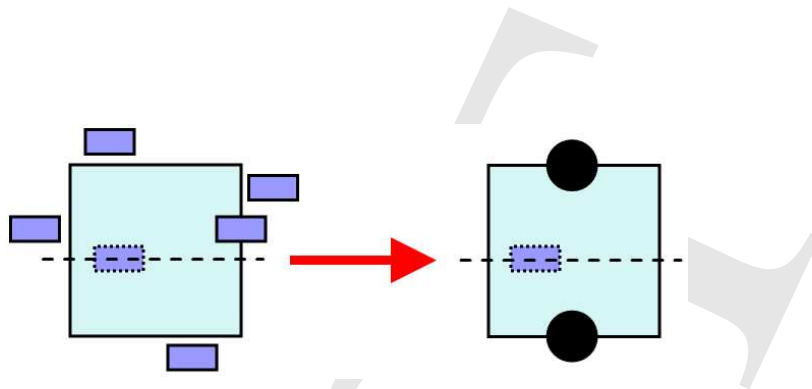


Figure 2: Terminal propagation. The net shown has five fixed terminals: four above and one below the cut-line. It also has movable cells which are represented by the cell with a dashed outline. The four fixed terminals above the cut-line are propagated to the black circle at the top of the bin while the one fixed terminal below the cut-line is propagated to the black circle below the cut-line. The movable cells remain unpropagated. Note that the net is *inessential* since terminals are propagated to both sides of the cut-line [32].

affect the quality of a placement since these connections can account for significant amounts of wirelength. On the other hand, these terminals are irrelevant to the classic partitioning formulation as they cannot be freely assigned to partitions. A compromise is possible by using an extended formulation of “partitioning with fixed terminals”, where the terminals are considered to be fixed in (“propagated to”) one or more partitions, and assigned zero areas (original areas are ignored). Nets which are propagated to both partitions in bi-partitioning are considered “inessential” since they will always be cut and can be safely removed from the partitioning instance to improve runtime [11]. Terminal propagation is typically driven by geometric proximity of terminals to subregions/partitions. Figure 2 depicts terminal propagation for a net with several fixed terminals. This particular net is inessential as it has terminals propagated to both sides of the cut-line.

2.2 Bipartitioning vs. Multi-way Partitioning

In his seminal work on min-cut placement, Breuer introduced two forms of recursive min-cut placement: slice/bisection and quadrature [7]. The style of min-cut placement most commonly used today has grown from the quadrature technique which advocated the use of horizontal and vertical cuts; the slice/bisection technique used only horizontal cuts and exhibited worse performance than quadrature [7].

Since that time, horizontal and vertical cut-lines have been standard in all placement techniques, but there has been debate as to whether there should be an ordering to the cuts (i.e. horizontally bisect a bin then vertically bisect its children as in quadrature [7]) or both cuts should be done simultaneously as in quadrisection [37]. Quadrisection has been shown to allow for the

optimization of techniques other than min-cut (such as minimal spanning tree length [21]), but terminal propagation is more complex when splitting a bin into four child bins instead of two. Also, bisection can simulate quadrisection with added flexibility in cut-line selection and shifting (see Section 2.3) [31]. There are currently no known methods that use greater than 4-way partitioning and the vast majority of partitioning-based placement techniques involve min-cut bipartitioning.

2.3 Cut-line Selection and Shifting

Breuer studied two types of cut-line direction selection techniques and found that alternating cut-line directions from layer to layer produced better half-perimeter wirelength (HPWL) than using only horizontal cuts [7]. The authors of [40] studied this phenomenon further by testing 64 cut-line direction sequences. Their experiments did not find that the two cut-sequences that alternate at each layer were the best, but did find that long sequences of cuts in the same direction during placement was detrimental to performance [40]. The authors of [44] developed a dynamic programming technique to choose optimal cut sequences for partitioning based placement, but also found that nearly optimal cut sequences could be determined from the aspect ratio of the bin to be split.

After cut-line direction is chosen, partitioning-based placers generally choose the cut-line that best splits a placement bin in half in the desired direction. Cut-lines are generally aligned to placement row and site boundaries to ease the assignment of standard-cells to rows near the end of global placement [9]. After a bin is partitioned, the initial cut-line may be moved, or shifted, in order to satisfy other objectives such as whitespace allocation or congestion reduction.

2.4 Whitespace Allocation

Management of whitespace (also known as free space) is a key issue in physical design as it has a profound effect on the quality of a placement. The amount of whitespace in a design is the difference between the total placeable area in a design and the total movable cell area in the design. A natural scheme for managing whitespace in top-down placement, uniform whitespace allocation, was introduced and analyzed in [12]. Let a placement bin which is going to be partitioned have *site area* S , *cell area* C , *absolute whitespace* $W = \max\{S - C, 0\}$ and *relative whitespace* $w = W/S$. A bi-partitioning divides the bin into two child bins with *site areas* S_0 and S_1 such that $S_0 + S_1 = S$ and *cell areas* C_0 and C_1 such that $C_0 + C_1 = C$. A partitioner is given cell area targets T_0 and T_1 as well as a tolerance τ for a particular bi-partitioning instance. In many cases of bi-partitioning,

$T_0 = T_1 = \frac{C}{2}$, but this is not always true [5]. τ defines the maximum percentage by which C_0 and C_1 are allowed to differ from T_0 and T_1 , respectively.

The work in [12] bases its whitespace allocation techniques on *whitespace deterioration*: the phenomenon that discreteness in partitioning and placement does not allow for exact uniform whitespace distribution. The whitespace deterioration for a bi-partitioning is the largest α , such that each child bin has at least αw relative whitespace. Assuming non-zero relative whitespace in the placement bin, α should be restricted such that $0 \leq \alpha \leq 1$ [12]. The authors note that $\alpha = 1$ may be overly restrictive in practice because it induces zero tolerance on the partitioning instance but $\alpha = 0$ may not be restrictive enough as it allows for child bins with zero whitespace, which can improve wirelength but impair routability [12].

For a given block, feasible ranges for partition capacities are uniquely determined by α . The partitioning tolerance τ for splitting a block with relative whitespace w is $\frac{(1-\alpha)w}{1-w}$ [12]. The challenge is to determine a proper value for α . First assume that a bin is to be partitioned horizontally n times more during the placement process. n can be calculated as $\lceil \log_2 R \rceil$ where R is the number of rows in the placement bin [12]. Assuming end-case bins have $\alpha = 0$ since they are not further partitioned, \bar{w} , the relative whitespace of an end-case bin, is determined to be $\frac{\bar{\tau}}{\bar{\tau}+1}$ where $\bar{\tau}$ is the tolerance of partitioning in the end-case bin [12].

Assuming that α remains the same during all partitioning of the given bin gives a simple derivation of $\alpha = \sqrt[n]{\frac{\bar{w}}{w}}$ [12]. A more practical calculation assumes instead that τ remains the same over all partitionings. This leads to $\tau = \sqrt[n]{\frac{1-\bar{w}}{1-w}} - 1$ [12]. \bar{w} can be eliminated from the equation for τ and a closed form for α based only w and n is derived to be $\alpha = \frac{n+1\sqrt{1-w}-(1-w)}{w(n+1\sqrt{1-w})}$ [12].

Free Cell Addition. One method of non-uniform whitespace allocation in placement, was presented in [3]. To achieve a non-uniform allocation of whitespace, free cells (standard cells that have no connections in the netlist) are added to the design which is then placed using uniform whitespace allocation. Care must be taken not to add too many cells to the design which can complicate the work of many placement algorithms, increasing interconnect length or leading to overlapping circuit modules [18].

Several other whitespace allocation techniques have been published in the literature, many of which have the objective of congestion reduction [28, 32, 38, 39, 43]. These techniques which deal specifically with congestion reduction are covered in a later chapter in this book ??.

3 Enhancements to the Min-cut Framework

This section describes several techniques which are recent improvements to the to the min-cut partitioning-based framework presented in Section 2. These techniques range from fairly simple yet effective techniques such as repartitioning and placement feedback to changes in the optimization goals of min-cut placement as in weighted net-cut.

3.1 Better Results Through Additional Partitioning

Huang and Kahng introduced two techniques for improving the results of quadrisection based placement known as cycling and overlapping [21]. Cycling is a technique whereby results are improved by partitioning every placement bin multiple times each layer [21]. After all bins are split for the first time in a layer of placement, a new round of partitioning on the same bins is done using the results of the previous round for terminal propagation. These additional rounds of partitioning are repeated until there is no further improvement of a cost function [21]. A similar type of technique was presented for min-cut bisection called *placement feedback*. In placement feedback, bins are partitioned multiple times, without requiring steady improvement in wirelength, to achieve more consistent terminal propagation [25].

Placement feedback serves to reduce the number of ambiguously propagated terminals. Ambiguity in terminal propagation arises when a terminal is nearly equidistant to the centers of the child bins of the bin being partitioned. In such cases it is unclear as to what side of the cut-line the terminal should be propagated. Traditional choices for such terminals is to propagate them to both sides or neither side of the cut-line in fear of making a poor decision [25]. Ambiguously propagated terminals introduce indeterminism into min-cut placement as they may be propagated differently based on the order in which placement bins are processed [25].

To reduce the number of ambiguously propagated terminals, placement feedback repeats each layer of partitioning n times. Each successive round of partitioning uses the resulting locations from the previous partitioning for terminal propagation. The first round of partitioning for a particular layer may have ambiguous terminals, but the second and later rounds will have reduced numbers of ambiguous terminals making terminal propagation more robust [25]. Empirical results show that placement feedback is effective in reducing HPWL, routed wirelength and via count [25].

The technique of overlapping also involves additional partitioning calls during placement [21].

While doing cycling in quadrisection, pieces of neighboring bins can be coalesced into a new bin and split to improve solution quality [21]. Brenner and Rohe introduced a similar technique which they called repartitioning which was designed to reduce congestion [6]. After partitioning, congestion was estimated in the placement bins of the design. Using this congestion data, new partitioning problems were formulated with all neighbors of a congested area. Solving these new partitioning problems would spread congestion to neighboring areas of the placement while possibly incurring an increase in net length [6].

Capo [30–35] repartitions bins similarly, but for the improvement of HPWL. After the initial solution of a partitioning problem is returned from a min-cut partitioner, Capo has the option of shifting the cut-line to fulfill whatever whitespace requirements may be asked of it. A shift of the cut-line, though, represents a change in the partitioning problem formulation (as the initial partitioning problem was built assuming a different cut-line which can have a significant effect on terminal propagation). Thus, the partitioning problem is rebuilt with the new cut-line location and solved again to improve wirelength. The repartitioning does not come with a significant runtime penalty because the initial partitioning solution is reused and modified by flat passes of a Fiduccia-Mattheyses [20] partitioner.

3.2 Fractional Cut

When a placement bin is split with a vertical cut-line, there are generally many possible cut-lines that can split the bin roughly equally since the size of sites in row-based placement is generally small. On the other hand, row heights are generally non-trivial as compared to the height of the core placement area. Since standard cells are ultimately placed in rows, most min-cut placers choose to align cut-lines to row boundaries [9]. The authors of [4] that this causes the “narrow region” problem which leads to instability in min-cut placement.

The “narrow region” problem becomes an issue when bins become tall and narrow. In such cases, total cell area may be able to fit into a given narrow bin, but it may not be possible to assign cells into these rows legally due to row area constraints or the number of legal solutions is so small that net-cut is artificially increased as a result [4]. Take for example a placement bin that encompasses two adjacent rows, each four units in length. If we have one cell with length five units and another with length two units, there is no way to legally place them in the rows, yet total

area constraints are satisfied.

To remedy this situation, the authors of [4] propose using a “fractional cut”: a horizontal cut-line that is allowed to pass through a fraction of a row. As horizontal cut-lines do not necessarily align with rows, cells must be assigned to rows before optimal end-case (typically single-row) placers can be used [4]. To legalize the placement, one proceeds on a row-by-row basis. Each cell is tentatively assigned to a preferred height in the placement: the center of its placement bin. Starting with the top-most row, cells are assigned to rows so as to minimize the cost of assigning cells. If a cell is assigned to the current row, its cost is the squared distance from its preferred position to the current row. If a cell is not assigned to the current row, its cost is the squared distance from its preferred position to the next lower row [4]. After all cells are assigned to rows, they are sorted by their x coordinates and packed in rows to remove any overlaps. The assignment of cells to rows is achieved efficiently by a dynamic programming formulation [4].

Experimental results show considerable improvements in terms of HPWL reduction in placement, but packing of cells in rows does not generally produce routable placements [32].

3.3 Analytical Constraint Generation

The authors of [5] note that min-cut placement techniques are effective at reducing HPWL of designs that are heavily constrained in terms of whitespace, but do not perform nearly as well as analytical techniques when there are large amounts of whitespace. The authors suggest that one reason for the discrepancy is that min-cut placers generally try to divide placement bins exactly in half with a relatively small tolerance. This tends to spread cell area roughly uniformly across the core area. Increasing the tolerance for partitioning a bin can allow for less uniformity in placement and lower HPWL due to tighter packing, but still does not reproduce the performance of analytical techniques [5].

To improve the HPWL performance of min-cut placement techniques on designs with large amounts of whitespace (which are becoming increasingly popular in real-world designs), while still retaining the good performance of min-cut techniques when there is limited whitespace, the authors of [5] suggest integrating analytical techniques and min-cut techniques. Before constructing a partitioning instance for a given placement bin, an analytical technique is run on the objects in the bin to minimize their quadratic wirelength [5]. Next, the center of mass of the placement of the

objects of the bin is calculated. This points to roughly where the objects should go to reduce their wirelength. Then one constructs a rectangle having the same aspect ratio as the placement bin and having the same area as the total area of movable objects in the bin. Let A be the total area of cells in the bin, H be the height of the bin and W the width of the bin. The height and width of such a rectangle can be calculated as follows: rectangle height $RH = \sqrt{\frac{A*H}{W}}$ and rectangle width $RW = \sqrt{\frac{A*W}{H}}$ [5]. One centers this rectangle at the center of mass of the analytical placement and intersects the rectangle with the proposed cut-line of the bin. The amount of area of the rectangle that falls on either side of the cut-line is used as a target for min-cut partitioning [5]. As most min-cut partitioners chose to split cell area equally, this is a significant departure from traditional min-cut placement.

Empirical results suggest that analytical constraint generation (ACG) is effective at improving the performance of min-cut placement on designs with large amounts of whitespace while retaining the good performance of min-cut placers on constrained designs while not impairing the routability of designs [5]. This performance comes at the cost of approximately 28% more runtime [5].

3.4 Better Modelling of HPWL by Partitioning

It is well-known that the min-cut objective in partitioning does not accurately represent the wirelength objective of placement [21, 36]. Optimizing HPWL and other objectives directly through partitioning can provide improvements over min-cut. Huang and Kahng showed that net weighting and quadrisection can be used to minimize a wide range of objectives such as minimal spanning tree cost [21]. Their technique consists of computing vectors of weights for each net (called net-vectors) and using these weights in quadrisection [21]. Although this technique can represent a wide range of cost functions to minimize, it requires the discretization of pin locations into the centers of bins and requires that sixteen weights must be calculated per net for partitioning [21].

The authors of [36] introduce a new terminal propagation technique in their placer THETO that allows the partitioner to better map net-cut to HPWL. The terminal propagation in THETO differs from traditional terminal propagation in that each original net may be represented by one or two nets in the partitioned netlist, depending on the configuration of the net's terminals. Two special cases — nets with no terminals and inessential nets — are treated the same as in traditional terminal propagation. Five other cases are analyzed in [36], based on the configuration of terminals relative

to the centers of the child bins, and proper weight computation is described (one case requires two nets). This way weighted net-cut better represents the “HPWL degradation” seen after partitioning. Empirically, this terminal propagation and net weighting are shown to reduce HPWL in min-cut placement.

This technique is simplified in [15] and reduced to the calculation of three wirelengths per net per partitioning instance (w_1 , w_2 and w_{12}) which completely determine the connectivity and costs of all nets in the derived partitioning hypergraph. While this formulation is more compact than that in [36], it is also more general. For each net in each partitioning instance, one must calculate the cost of all nodes on the net being placed in partition 1 (w_1), the cost of all nodes on the net being placed in partition 2 (w_2) and the cost of all nodes on the net being split between partitions 1 and 2 (w_{12}). Up to two nets can be created in the partitioning instance, one with weight $|w_1 - w_2|$ and the other with weight $w_{12} - \max(w_1, w_2)$. The only assumption made in [15] is that $w_{12} \geq \max(w_1, w_2)$. With these costs and the corresponding connectivity of the derived hypergraph, minimizing weighted net-cut directly corresponds to minimizing HPWL.

4 Mixed-size Placement

Mixed-size placement, the placement of large macros in addition to standard cells, has become a relevant challenge in physical design and is poised to dominate physical design in the near future as we move from traditional “sea of cells” ICs to “sea of hard macros” SoCs [42]. To keep up with this shift in physical design, several techniques for partitioning based mixed-size placement have been proposed and are described in this section. These techniques include floorplacement, PATOMA, and mixed-size placement with fractional cut.

4.1 Floorplacement

From an optimization point of view, floorplanning and placement are very similar problems – both seek non-overlapping placements to minimize wirelength. They are mostly distinguished by scale and the need to account for shapes in floorplanning, which calls for different optimization techniques. Netlist partitioning is often used in placement algorithms, where geometric shapes of partitions can be adjusted. This considerably blurs the separation between partitioning, placement and floorplanning, raising the possibility that these three steps can be performed by one CAD tool.

```

Variables: queue of placement bins
Initialize queue with top-level placement bin
1  While (queue not empty)
2    Dequeue a bin
3    If (bin has large/many macros or is marked as merged)
4    Cluster std-cells into soft macros
5    Use fixed-outline floorplanner to pack
6    all macros (soft+hard)
7    If fixed-outline floorplanning succeeds
8    Fix macros and remove sites underneath the macros
9    Else
10   Undo one partition decision. Merge bin with sibling
11   Mark new bin as merged and enqueue
12 Else if (bin small enough)
13   Process end case
14 Else
15   Bi-partition the bin into smaller bins
    Enqueue each child bin

```

Figure 3: Min-cut floorplacement. Bold-faced lines 3-10 are different from traditional min-cut placement [31].

The authors of [31] develop such a tool and term the unified layout optimization *floorplacement* following Steve Teig’s keynote speech at ISPD 2002.

Min-cut placers scale well in terms of runtime and wirelength minimization, but cannot produce non-overlapping placements of modules with a wide variety of sizes. On the other hand, annealing-based floorplanners can handle vastly different module shapes and sizes, but only for relatively few (100-200) modules at a time. Otherwise, either solutions will be poor or optimization will take too long to be practical. The loose integration of fixed-outline floorplanning and standard-cell placement proposed in [2] suffers from a similar drawback because its single top-level floorplanning step may have to operate on numerous modules. Bottom-up clustering can improve the scalability of annealing, but not sufficiently to make it competitive with other approaches. The work in [31] applies min-cut placement as much as possible and delays explicit floorplanning until it becomes necessary. In particular, since min-cut placement generates a slicing floorplan, it is viewed as an implicit floorplanning step, reserving explicit floorplanning for “local” non-slicing block packing.

Placement starts with a single placement bin representing the entire layout region with all the placeable objects initialized at the center of the bin. Using min-cut partitioning, the bin is split into two bins of similar sizes, and during this process the cut-line is adjusted according to actual partition sizes. Applying this technique recursively to bins (with terminal propagation) produces a series of gradually refined slicing floorplans of the entire layout region. In very small bins, all cells can be placed by a branch-and-bound end-case placer [11]. However, this scheme breaks down

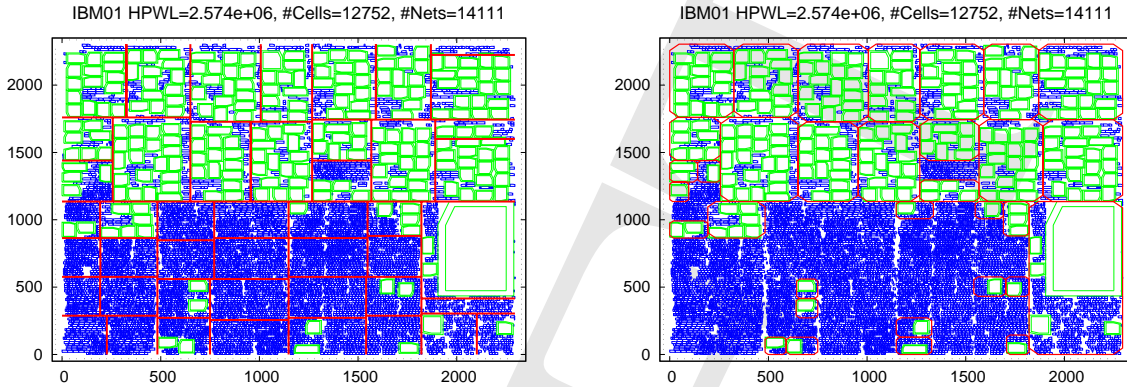


Figure 4: Progress of mixed-size floorplacement on the IBM01 benchmark from IBM-MSwPins. The picture on the left shows how the cut lines are chosen during the first six layers of min-cut bisection. On the right is the same placement but with the floorplanning instances highlighted by “rounded” rectangles. Floorplanning failures can be detected by observing nested rectangles [31].

on modules that are larger than their bins. When such a module appears in a bin, recursive bisection cannot continue, or else will likely produce a placement with overlapping modules. Indeed, the work in [27] continues bisection and resolves resulting overlaps later. In this technique, one switches from recursive bisection to “local” floorplanning where the fixed outline is determined by the bin. This is done for two main reasons: (i) to preserve wirelength [8], congestion [6] and delay [23] estimates that may have been performed early during top-down placement, and (ii) avoid legalizing a placement with overlapping macros.

While deferring to fixed-outline floorplanning is a natural step, successful fixed-outline floorplanners have appeared only recently [1]. Additionally, the floorplanner may fail to pack all modules within the bin without overlaps. As with any constraint-satisfaction problem, this can be for two reasons: either (i) the instance is unsatisfiable, or (ii) the solver is unable to find any of existing solutions. In this case, the technique undoes the previous partitioning step and merges the failed bin with its sibling bin, whether the sibling has been processed or not, then discards the two bins. The merged bin includes all modules contained in the two smaller bins, and its rectangular outline is the union of the two rectangular outlines. This bin is floorplanned, and in the case of failure can be merged with its sibling again. The overall process is summarized in Figure 3 and an example is depicted in Figure 4.

It is typically easier to satisfy the outline of a merged bin because circuit modules become

relatively smaller. However, Simulated Annealing takes longer on larger bins and is less successful in minimizing wirelength. Therefore, it is important to floorplan at just the right time, and the algorithm determines this point by backtracking. Backtracking does incur some overhead in failed floorplan runs, but this overhead is tolerable because merged bins take considerably longer to floorplan. Furthermore, this overhead can be moderated somewhat by careful prediction.

For a given bin, a floorplanning 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 conglomerating standard cells into soft placeable blocks. This is accomplished by a simple bottom-up connectivity-based clustering [26]. The existing large modules in the bin are usually kept out of this clustering. To further simplify floorplanning, soft blocks consisting of standard cells are artificially downsized, as in [3]. The clustered netlist is then passed to the fixed-outline floorplanner Parquet [1], which sizes soft blocks and optimizes block orientations. After suitable locations are found, the locations of all large modules are returned to the top-down placer and are considered fixed. The rows below those modules are fractured and their sites are removed, i.e., the modules are treated as fixed obstacles. At this point, min-cut placement resumes with a bin that has no large modules in it, but has somewhat non-uniform row structure. When min-cut placement is finished, large modules do not overlap by construction, but small cells sometimes overlap (typically below 0.01% by area). Those overlaps are quickly detected and removed with local changes.

Since the floorplacer includes a state-of-the-art floorplanner, it can natively handle pure block-based designs. Unlike most algorithms designed for mixed-size placement, it can pack blocks into a tight outline, optimize block orientations and tune aspect ratios of soft blocks. When the number of blocks is very small, the algorithm applies floorplanning quickly. However, when given a larger design, it may start with partitioning and then call fixed-outline floorplanning for separate bins. As recursive bisection scales well and is more successful at minimizing wirelength than annealing-based floorplanning, the proposed approach is scalable and effective at minimizing wirelength.

Empirical boundary between placement and floorplanning. By identifying the characteristics of placement bins for which the algorithm calls floorplanning, one can tabulate the empirical boundary between placement and floorplanning. Formulating such *ad hoc* thresholds in terms of

| Floorplanning conditions for floorplacement | |
|---|--|
| N, n : | The numbers of large modules and movable objects in a given bin. |
| $A(m)$: | The area of the m largest modules in a given bin, $m \leq n$. |
| C : | The capacity of a given bin. |
| Test 1. | At least one large module does not fit into a potential child bin. |
| Test 2. | $N \leq 30$ and $A(N) < 0.80 * A(n)$ and $A(n) > 0.6 * C$. |
| Test 3. | $N \leq 15$ and $A(N) < 0.95 * A(n)$ and $A(n) > 0.6 * C$. |
| Test 4. | $A(50) < 0.85 * C$. |
| Test 5. | $A(10) < 0.60 * C$. |
| Test 6. | $A(1) < 0.30 * C$ and $N = 1$. |
| Test 7. | $N = n = 1$. |

Table 1: Floorplanning conditions used in floorplacement [31]. Test 1 is the most fundamental, for if a bin meeting test 1 were not floorplanned, a failure would be guaranteed at the next level. Tests 2-6 detect bins dominated by large macros. Test 7 is a base case where only one module exists, but it is large.

dimensions of the largest module in the bin, etc., allows one to avoid unnecessary backtracking and decrease the overhead of floorplanning calls that fail to satisfy the fixed outline constraint because they are issued too late. In practice, issuing floorplanning calls too early (i.e., on larger bins) increases final wirelength and sometimes runtime. To improve wirelength, the *ad hoc* tests for large modules in bins (that trigger floorplanning) are deliberately conservative.

These conditions were derived by closely monitoring the legality of floorplanning and min-cut placement solutions. When a partitioned bin yields an illegal placement solution it is clear that the bin should have been floorplanned and a condition should be derived. When a call to floorplanning fails to satisfy the fixed outline constraint the placer has to backtrack. To avoid paying this penalty, a condition should be derived to allow for floorplanning the parent bin and prevent the failure.

These conditions are refined to prevent floorplanning failure by visual inspection of a plot of the resulting parent bin and formulating a condition describing its composition. An example of such a plot is shown in Figure 4. Floorplanned bins are outlined with rounded rectangles. Nested rectangles indicate a failed floorplan run, followed by backtracking and floorplanning of the larger parent bin. In our experience, these tests are strong enough to ensure that at most one level of backtracking is required to prevent overlaps between large modules.

4.2 PATOMA and PolarBear

PATOMA 1.0 [17] pioneered a top-down floorplanning framework that utilizes fast block-packing algorithms (ROB or ZDS [16]) and hypergraph partitioning with hMETIS [26]. This approach is fast and scalable, and provides good solutions for many input configurations. Fast block-packing is used in PATOMA to guarantee that a legal packing solution exists, at which point the burden of wirelength minimization is shifted to the hypergraph partitioner. This idea is applied recursively to each of the newly-created partitions. In end-cases, when a partitioning step leads to unsatisfiable block-packing, the quality of the result is determined by the quality of its fast block-packing algorithms. In end-cases, when partitioning cannot be used because it creates unsatisfiable instances of block-packing, block locations are determined by fast block-packing heuristics. The placer PolarBear [18] integrates algorithms from PATOMA to increase the robustness of a top-down min-cut placement flow. Similar to PATOMA, the floorplanner IMF [15] utilizes top-down partitioning, but allows overlaps in the initial top-down partitioning phase. A bottom-up merging and refinement phase fixes overlaps and further optimizes the solution quality.

4.3 Fractional Cut for Mixed-size Placement

The work in [27] advocates a two-stage approach to mixed-size placement. First, the min-cut placer FengShui [4] generates an initial placement for the mixed-size netlist without trying to prevent all overlaps between modules. The placer only tracks the global distribution of area during partitioning and uses the *fractional cut* technique (see Section 3.2), which further relaxes book-keeping by not requiring placement bins to align to cell rows. While giving min-cut partitioners more freedom, these relaxations prevent cells from being placed in rows easily and require additional repair during detail placement. This may particularly complicate the optimization of module orientations, not considered in [27] (relevant benchmarks use only square blocks with all pins placed in the centers).

The second stage consists of removing overlaps by a fast legalizer designed to handle large modules along with standard cells. The legalizer is essentially greedy and attempts to shift all modules towards the left edge of the chip (or to the right edge, if that produces better results). In our experience, the implementation reported in [27] leads to horizontal stacking of modules and sometimes yields out-of-core placements, especially when several very large modules are present

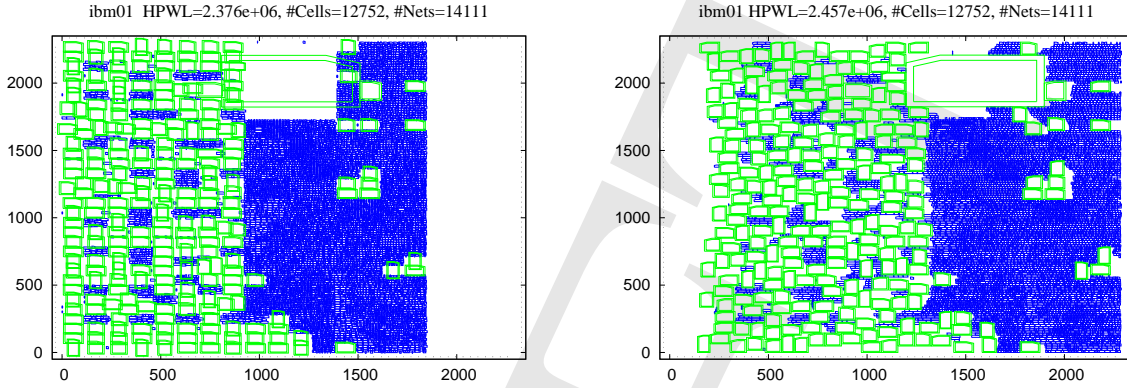


Figure 5: A placement of the IBM01 benchmark from IBM-MSwPins by FengShui before (left) and after (right) legalization and detail placement.

(the benchmarks used in [27] contain numerous modules of medium size). See Figure 10 in [31] and Figure 6 in [30] for examples of this behavior. Another concern about packed placements is the harmful effect of such a strategy on routability, explicitly shown in [43]. Overall, the work in [27] demonstrates very good legal placements for common benchmarks, but questions remain about the robustness and generality of the proposed approach to mixed-size placement. Example FengShui placements before and after legalization are shown in Figure 5.

5 Advantages of Min-cut Placement

This section presents recent techniques which give min-cut placement a significant advantage over other placement algorithms in whitespace allocation, floorplacement, routed wirelength and incremental placement.

5.1 Flexible Whitespace Allocation

The min-cut bisection based placement framework offers much flexibility in whitespace allocation. Section 2.4 describes uniform allocation of whitespace for min-cut bisection placement and a trivial pre-processing step to allow for non-uniform allocation. This section outlines two more sophisticated whitespace allocation techniques, minimum local whitespace and safe whitespace, that can be used for non-uniform whitespace allocation and satisfying whitespace constraints [35].

Minimum Local Whitespace. If a placement bin has more than a user-defined minimum local whitespace (`minLocalWS`), partitioning will define a tentative cut-line that divides the bin's

placement area in half. Partitioning targets an equal division of cell area, but is given more freedom to deviate from its target. Tolerance is computed so that with whitespace deterioration, each descendant bin of the current bin will have at least `minLocalWS` [35].

The assumption that the whitespace deterioration, α , in end-case bins is 0 made in [12] and presented in Section 2.4 no longer applies, so the calculation of α must change. Since we want all child bins of the current bin to have `minLocalWS` relative whitespace, in particular end case bins must have at least `minLocalWS` and thus we may set $\bar{w} = \text{minLocalWS}$, instead of a function of τ . Using the assumption that α remain constant during partitioning, α can be calculated directly as $\alpha = \sqrt[n]{\frac{\bar{w}}{w}}$ [12]. With the more realistic assumption that τ remain constant, τ can be calculated as $\tau = \sqrt[n]{\frac{1-\bar{w}}{1-w}} - 1$ [12]. Knowing τ , α can be computed as $\alpha = (\tau + 1) + \frac{\tau}{w}$ [12].

After a partitionment is calculated, the cut-line is shifted to ensure that `minLocalWS` is preserved on both sides of the cut-line. If the minimum local whitespace is chosen to be small, one can produce tightly packed placements which greatly improves wirelength.

Safe Whitespace. The last whitespace allocation mode is designed for bins with “large” quantities of whitespace. In safe whitespace allocation, as with minimum local whitespace allocation, a tentative geometric cut-line of the bin is chosen, and the target of partitioning is an equal bisection of the cell area. The difference in safe whitespace allocation mode is that the partitioning tolerance is much higher. Essentially, any partitioning solution that leaves at least `safeWS` on either side of the cut-line is considered legal. This allows for very tight packing and reduces wirelength, but is not recommended for congestion-driven placement [35].

Figure 6 illustrates uniform and non-uniform whitespace allocation. Column (a) shows global placements with uniform (top) and non-uniform (bottom) whitespace allocation on the ISPD 2005 contest benchmark `adaptecl` (57.34% utilization) [29]. In the non-uniform placement shown, the minimum local whitespace is 12% and safe whitespace is 14%. Columns (b) and (c) show intensity maps of the local utilization of each placement. Lighter areas of the intensity maps signify violations of a given target placement density; darker areas have utilization below the target. Regions completely occupied by fixed obstacles are shaded as if they exactly meet the target density. The target densities for columns (b) and (c) are 90% and 60%. Note that uniform whitespace produces almost no violations when the target is 90% and relatively few when the target is 60%. The non-uniform placement has more violations as compared to the uniform placement especially when the

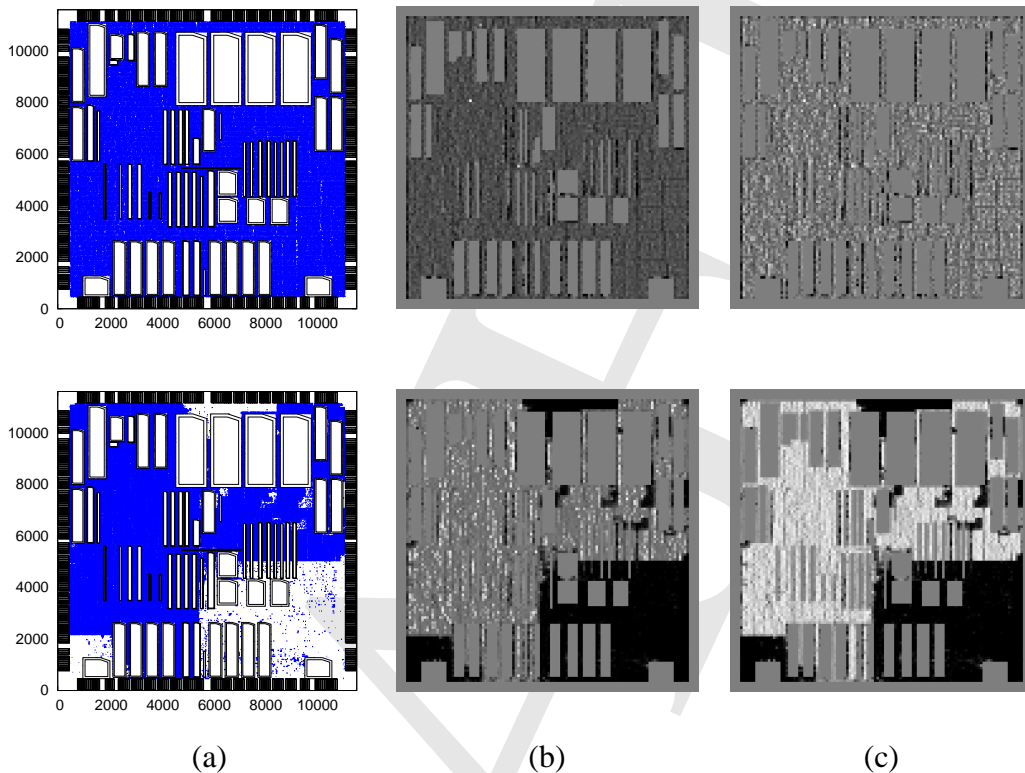


Figure 6: Column (a) shows global placements of the ISPD 2005 Placement contest benchmark adaptec1 [29] (57.34% utilization) with uniform whitespace allocation (top) and non-uniform whitespace allocation (bottom). Fixed obstacles are drawn with double lines. To indicate orientation, north-west corners of blocks are truncated. Columns (b) and (c) depict the local utilization of the placements. Lighter areas of the placement signify placement regions with density above a given target (90% for column (b) and 60% for column (c)) whereas darker areas have utilization below the target.

target is 60%, but remains largely legal with the 90% target density.

5.2 Solving Difficult Instances of Floorplacement

Floorplacement (see Section 4.1) appears promising for SoC layout because of its high capacity and the ability to pack blocks. However, as experiments in [30] demonstrate, existing tools for floorplacement are fragile — on many instances they fail, or produce remarkably poor placements.

To improve the performance of min-cut placement on mixed-size instances, the authors of [30] propose three synergistic techniques for floorplacement that in particular succeed on hard instances: (i) selective floorplanning with macro clustering, (ii) improved obstacle evasion for B*-tree, and (iii) ad hoc look-ahead in top-down floorplacement. Obstacle evasion is especially impor-

```

Variables: queue of placement partitions
Initialize queue with top-level partition
1  While (queue not empty)
2    Dequeue a partition
3    If (partition is not marked as merged)
4      Perform look-ahead floorplanning on partition
5      If look-ahead floorplanning fails
6        Undo one partition decision
7        Merge partition with sibling
8        Mark new partition as merged and enqueue
9    Else if (partition has large macros or
            is marked as merged)
10     Mark large macros for placement after floorplanning
11     Cluster remaining macros into soft macros
12     Cluster std-cells into soft macros
13     Use fixed-outline floorplanner to pack
        all macros (soft+hard)
14     If fixed-outline floorplanning succeeds
15       Fix large macros and remove sites beneath
16     Else
17       Undo one partition decision
18       Merge partition with sibling
19       Mark new partition as merged and enqueue
20     Else if (partition is small enough and
        mostly comprised of macros)
21     Process floorplanning on all macros
22     Else if (partition small enough)
23       Process end case std cell placement
24     Else
25       Bi-partition netlist of the partition
26       Divide the partition by placing a cutline
27       Enqueue each child partition

```

Figure 7: Modified min-cut floorplacement flow. Bold-faced lines are new [30].

tant for top-down floorplacement, even for designs that initially have no obstacles. The techniques are called SCAMPI, an acronym for *SCalable Advanced Macro Placement Improvements*. Empirically, SCAMPI shows significant improvements in floorplacement success rate (68% improvement as compared to the floorplacement technique presented in Section 4.1) and HPWL (3.5% reduction compared to floorplacement in Section 4.1).

Traditional placement techniques such as top-down and analytical frameworks, bottom-up clustering and iterative cell-spreading, scale well in terms of runtime and interconnect optimization *when all modules are small*. However, handling a wide variety of module sizes with these techniques seems considerably more difficult [30]. On the other hand, simulated annealing has a good track record in handling heterogeneous module configurations, but can only be effectively applied to small problem sizes [30]. This dichotomy between large-scale placement techniques and annealing-based floorplanning necessitates a rethinking of existing floorplacement flows [30].

Selective floorplanning with macro clustering. In top-down correct-by-construction frame-

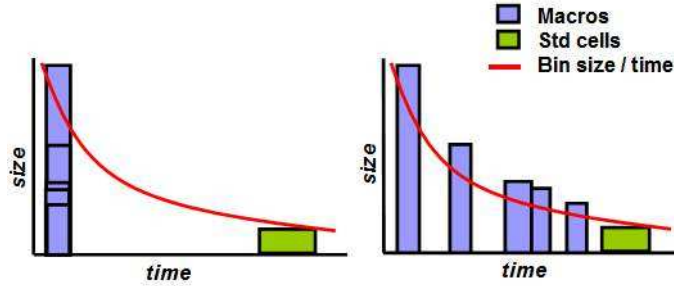


Figure 8: The plot on the left illustrates traditional floorplacement. Whenever a floorplanning threshold is reached, all macros in the bin are designated for floorplanning. Then, the floorplacement flow continues down until detailed placement, where the standard cells will be placed. The plot on the right illustrates the SCAMPI flow. Macros are selectively placed at the appropriate levels of hierarchy [30].

works like Capo and PATOMA [17] (see Section 4.2), a key bottleneck is in ensuring ongoing progress — partitioning, floorplanning or end-case processing must succeed at any given step. Both frameworks experience problems when floorplanning is invoked too early to produce reasonable solutions — PATOMA resorts to solutions with very high wirelength, and Capo times out because it has nothing to resort to and runs the an annealer on too many modules. In order to scale better, the annealer clusters small standard cells into soft blocks before starting Simulated Annealing. When a solution is available, all hard blocks are considered placed and fixed — they are treated as obstacles when the remaining standard cells are placed. Compared to other multi-level frameworks, this one does not include refinement, which makes it relatively fast. Speed is achieved at the cost of not being able to cluster modules other than standard cells because the floorplanner does not produce locations for clustered modules. Unfortunately, this limitation significantly restricts scalability of designs with many macros [30].

The proposed technique of *selective floorplanning with macro clustering* allows to cluster blocks before annealing, and does not require additional refinement or cluster-packing steps (which are among the obvious facilitators) — instead certain existing steps in floorplacement are skipped. This improvement is based on two observations: (i) blocks that are much smaller than their bin can be treated like standard cells, (ii) the number of blocks that are large relative to the bin size is necessarily limited. E.g., there cannot be more than nine blocks with area in excess of 10% of a bin’s area [30].

In selective floorplanning, each block is marked as small or large based on a size threshold.

Standard cells and small blocks can be clustered, except that clusters containing hard blocks have additional restrictions on their aspect ratios. After successful annealing, only the large blocks are placed, fixed and considered obstacles. Normal top-down partitioning resumes, and each remaining block will qualify as large at some point later. This way, specific locations are determined when the right level of detail is considered. If floorplanning fails during hierarchical placement, the failed bin is merged with its sibling and the merged bin is floorplanned (see Figure 7). The blocks marked as large in the merged bin include those that exceed the size threshold and also those marked as large in the failed bin (since the failure suggests that those blocks were difficult to pack). After the largest macros are placed, the flow resumes [30].

The proposed technique limits the size of floorplanning instances given to the annealer *by a constant* (in our case 200 modules) and does not require much extra work. However, it introduces an unexpected complexity. The floorplacement framework does not handle fixed obstacles in the core region, and none of the public benchmarks have them. When Capo fixes blocks in a particular bin, it fixes *all of them* and never needs to floorplan around obstacles. Another complication due to newly introduced fixed obstacles is in outline selection. Reliable obstacle-evasion and intelligent outline selection may be required by practical designs, even without selective floorplanning (e.g., to handle pre-diffused memories, built-in multipliers in FPGAs, etc). Therefore they are viewed as independent but synergistic techniques [30].

Obstacle evasion in floorplanning: B*-tree enhancement. When satisfying area constraints is difficult, it is very important to increase the priority of area optimization so as to achieve legality [14]. Because of this, the authors of [30] select the B*-tree [13] floorplan representation for its amenability to packed configurations and add obstacle evasion into B*-tree evaluation.

Ad-hoc look-ahead floorplanning. The sum of block areas may significantly under-estimate the area required for large blocks. Better estimates are required to improve the robustness of floorplacement and look-ahead area-driven floorplanning appears as a viable approach [30].

SCAMPI performs look-ahead floorplanning to validate solutions produced by the hypergraph partitioner, and check that a resulting partition is packable, within a certain tolerance for failure. Look-ahead floorplanning must be fast, so that the amortized runtime overhead of the look-ahead calls is less than the total time saved from discovering bad partitioning solutions. Therefore look-ahead floorplanning is performed with blocks whose area is larger than 10% of the total module

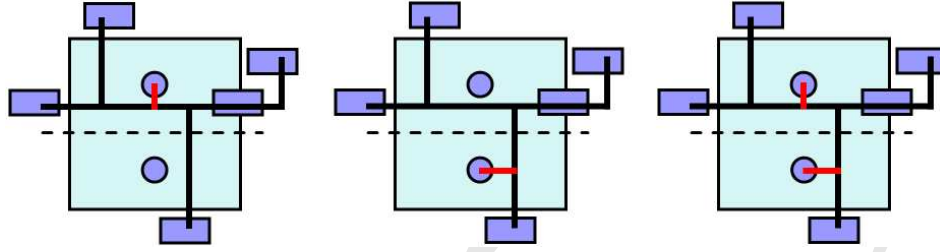


Figure 9: Calculating the three costs for weighted terminal propagation with StWL: w_1 (left), w_2 (middle), and w_{12} (right). The net has five fixed terminals: four above and one below the proposed cut-line. For the traditional HPWL objective, this net would be considered inessential. Note that the structure of the three Steiner trees may be entirely different, which is why w_1 , w_2 and w_{12} are evaluated independently [32].

area in the bin, and soft blocks containing remaining modules. For speed, the floorplanner is configured to perform area-only packing, and the placer is configured to only perform look-ahead floorplanning on bins with large blocks. Dealing with only the largest blocks is sufficient because floorplanning failures are most often caused by such blocks [30].

5.3 Optimizing Steiner Wirelength

Weighted terminal propagation as described in [15] and summarized in Section 3.4, is sufficiently general to account for objectives other than HPWL such as Steiner Wirelength (StWL) [32]. StWL is known to correlate with final routed wirelength (rWL) more accurately than HPWL and the authors of [32] hypothesize that if StWL could be directly optimized during global placement, one may be able to enhance routability and reduce routed wirelength.

The points required to calculate w_1 for a given net are the terminals on the net plus the center of partition 1. Similarly, the points required to calculate w_2 are the terminals plus the center of partition 2. Lastly, the points to calculate w_{12} are the terminals on the net plus the centers of both partitions. See Figure 9 for an example of calculating these three costs. Clearly the HPWL of the set of points necessary to calculate w_{12} is at least as large as that of w_1 and w_2 since it contains an additional point. By the same logic, StWL also satisfies this relationship since RSMT length can only increase with additional points. Since StWL is a valid cost function for these weighted partitioning problems, this is a framework whereby it can be minimized [32].

The simplicity of this framework for minimizing StWL is deceiving. In particular, the propagation of terminal locations to the current placement bin and the removal of inessential nets [11]

— standard techniques for HPWL minimization — cannot be used when minimizing StWL. Moving terminal locations drastically changes Steiner-tree construction and can make StWL estimates extremely inaccurate. Nets that are considered inessential in HPWL minimization (where the x- or y-span of terminals, if the cut is vertical or horizontal respectively, contains the x- or y-span of the centers of child bins) are not necessarily inessential when considering StWL because there are many Steiner trees of different lengths that have the same bounding box. Figure 9 illustrates a net that is inessential for HPWL minimization but essential for StWL minimization. Not only computing Steiner trees, but even traversing all relevant nets to collect all relevant point locations can be very time-consuming. Therefore, the main challenge in supporting StWL minimization is to develop efficient data structures and limit additional runtime during placement [32].

Pointsets with multiplicities. Building Steiner trees for each net during partitioning is a computationally expensive task. To keep runtime reasonable when building Steiner trees for partitioning, the authors of [32] introduce a simple yet highly effective data structure — *pointsets with multiplicities*. For each net in the hypergraph, two lists are maintained. The first list contains all the unique pin locations on the net that are fixed. A fixed pin can come from sources such as terminals or fixed objects in the core area. The second list contains all the unique pin locations on the net that are movable, i.e., all other pins that are not on the fixed list. All points on each list are unique so that redundant points are not given to Steiner evaluators which may increase their runtime. To do so efficiently, the lists are kept in a sorted order. For both lists, in addition to the location of the pin, the number of pins that correspond to a given point is also saved [32].

Maintaining the number of actual pins that correspond to a point in a pointset (the multiplicity of that point) is necessary for efficient update of pin locations during placement. If a pin changes position during placement, the pointsets for the net connected to the pin must be updated. First, the original position of the pin must be removed from the movable point set. As multiple pins can have the same position, especially early in placement, the entire net would need to be traversed to see if any other pins share the same position as the pin that is moving. Multiplicities allow to know this information in constant time. To remove the pin, one performs a binary search on the pointset and decreases the multiplicity of the pin’s position by 1. If this results in the position having a multiplicity of 0, the position can be removed entirely. Insertion of the pin’s new position is similar: first, a binary search is performed on the pointset. If the pin’s position is already present

in the pointset, the multiplicity is increased by 1. Otherwise, the position is added in sorted order with a multiplicity of 1. Empirically, building and maintaining the pointset data structures takes less than 1% of the runtime of global placement [32].

Performance. The authors of [32] compared three Steiner evaluators in terms of runtime impact and solution quality. They chose the FastSteiner [24] evaluator for global placement based on its reasonable runtime and consistent performance on large nets. Empirical results show the use of FastSteiner leads to a reduction of StWL by 3% on average on the IBMv2 benchmarks [43] (with a reduction of routed wirelength up to 7%) while using less than 30% additional runtime [32].

5.4 Incremental Placement

To develop a strong incremental placement tool, ECO-system, the authors of [33] build upon an existing global placement framework and must choose between analytical and top-down. The main considerations include robustness, the handling of movable macros and fixed obstacles, as well as consistent routability of placements and the handling of density constraints. Based on recent empirical evidence [30,32,35], the top-down framework appears a somewhat better choice. However, analytical algorithms can also be integrated into ECO-system when particularly extensive changes are required. ECO-system favorably compares to recent detail placers in runtime and solution quality and fares well in high-level and physical synthesis.

General Framework. ECO-system can be likened to reverse engineering the min-cut placement process. The goal is to reconstruct the internal state of a min-cut placer that could have produced the given initial placement. Given this state, one can choose to accept or reject its previous decisions based on their own criteria and build a new placement for the design. If many of the decisions of the placer were good, one can achieve a considerable runtime savings as compared to placement from scratch. If many of the decisions are determined to be bad, one can do no worse in terms of solution quality than placement from scratch. An overview of the application of ECO-system to an illegal placement is depicted in Figure 11. The overall algorithm in the framework of min-cut placement is shown in Figure 10.

To rebuild the state of a min-cut placer, one must reconstruct a series of cut-lines and partitioning solutions efficiently. One must also determine criteria for the acceptability of the derived partitioning and cut-line. To extract a cut-line and partitioning solution from a given placement bin,


```

Variables: queue of placement bins
Initialize queue with top-level placement bin
1  While(queue not empty)
2    Dequeue a bin
3    If(bin not marked to place from scratch)
4    If(bin overfull)
5    Mark bin to place from scratch, break
6    Quickly choose the cut-line which has
    the smallest net-cut considering
    cell area balance constraints
7    If(cut-line causes overfull child bin)
8    Mark bin to place from scratch, break
9    Induce partitioning of bin's cells from cut-line
10   Improve net-cut of partitioning with
    single pass of Fiduccia-Mattheyses
11   If(% of improvement > threshold)
12   Mark bin to place from scratch, break
13   Create child bins using cut-line and partitioning
14   Enqueue each child bin
15   If(bin marked to place from scratch)
16     If(bin small enough)
17       Process end case
18     Else
19       Bi-partition the bin into child bins
20       Mark child bins to place from scratch
21       Enqueue each child bin

```

Figure 10: Incremental min-cut placement. Bold-faced lines 3-15 and 20 are different from traditional min-cut placement [33].

all possible cut-lines of the bin as well as the partitions they induce must be examined. Starting at one edge of the placement bin (left edge for a vertical cut and bottom edge for a horizontal cut) and moving towards the opposite edge, for each potential cut-line encountered, one maintains the cell area on either side of the cut-line, the partition induced by the cut-line and its net cut.

Once a cut-line and partitioning have been chosen, they must be evaluated to see if they should be accepted or rejected. To evaluate the partitioning, the authors of [33] use it as input to a Fiduccia-Mattheyses partitioner and see how much it can be improved by a single pass (if the bin is large enough, a multi-level Fiduccia-Mattheyses partitioner can be used). The intuition is that if the constructed partitioning is not worthy of reuse, a single Fiduccia-Mattheyses pass could improve its cut non-trivially. If the Fiduccia-Mattheyses pass improves the cut beyond a certain threshold, the solution is discarded and the entire bin is bisected from scratch. If a partition is accepted by this criterion, one performs a legality test: if the partitioning overfills a child bin, the cut-line is discarded and the bin is bisected from scratch.

Empirically, the runtime of the cut-line selection procedure (which includes a single pass of a Fiduccia-Mattheyses partitioner) is much smaller than partitioning from scratch. On large bench-

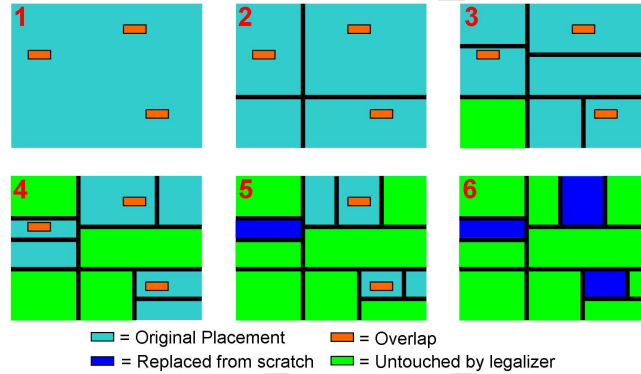


Figure 11: Legalization during min-cut placement. Placement bins are subdivided until (i) a bin contains no overlap and is ignored for the remainder of the legalization process or, (ii) the placement contained in the bin is considered too poor to be kept (too many overlaps or does not meet the solution quality requirements) and is replaced from scratch using min-cut or analytical techniques [33].

marks, the cut-line selection process requires 5% of ECO-system runtime time whereas min-cut partitioners generally require 50% or more of ECO-system runtime. ECO-system as a whole requires approximately 15% of original placement runtime.

Handling Macros and Obstacles. With the addition of macros, the flow of top-down placement usually becomes more complex. The authors of [33] adopt the style of floorplacement from [30, 31] (see Sections 4.1 and 5.2). For legalization with macros, a new criterion for floorplanning is added: if a placement bin has non-overlapping positions for macros (i.e. no macros in the placement bin overlap each other) the macros are placed in exactly their initial positions; if some of the macros overlap, other floorplanning criteria are used to decide. If any of the macros are moved, the placement of all cells and macros in the bin must be discarded and placement proceeds as described in [31].

During the cut-line selection process, some cut-line locations are considered invalid — namely those that are too close to obstacle boundaries but do not cross the obstacles. This is done to prevent long and narrow slivers of space between cut-lines and obstacle boundaries. Ties for cut-lines are broken based on the number of macros they intersect. This helps to reduce overfullness in child bins allowing deeper partitioning, which reduces runtime [33].

6 State-of-the-art Min-cut Placers

In this section, we present partitioning-based placement techniques that are used in cutting-edge placers. For each placer, we describe its overall flow, how this differs from the generic min-cut flow, and how it handles challenges in placement such as fixed obstacles and mixed-size instances. In particular we describe the techniques used by the placers Dragon [38, 39, 43], FengShui [4, 27], NTUPlace2 [22] and Capo [30–35].

6.1 Dragon

The most recent version of Dragon, Dragon2006 [39], combines min-cut bisection with Simulated Annealing for placement. In its most basic flow, Dragon2006 utilizes recursive bisection with the hMETIS partitioner [26]. Each bin is partitioned multiple times with a feedback mechanism to allow for more accurate terminal propagation (see Section 3.1 for more details on placement feedback). Partitioning is followed by Simulated Annealing on the placement bins where whole bins are swapped with one another to improve HPWL [38, 39]. After a number of layers of interleaved partitioning and Simulated Annealing, each bin contains only a few cells and the partitioning phase terminates. Next, bins are aligned to row structures and cell-based Simulated Annealing is performed wherein cells are swapped between bins to improve HPWL [38, 39]. Lastly, cell overlaps are removed and local detail placement improvements are made.

Mixed-size Placement. The traditional Dragon flow does not take macros into consideration during placement. To account for macros, partitioning, bin-based annealing and legalization must be modified. In addition, Dragon2006 makes two passes on a design with obstacles; the first pass finds locations for macros and the second treats macros as fixed obstacles [39] (similar to [2]).

In the first pass, partitioning is modified to handle large movable macros. The traditional Dragon flow alternates cut directions at each layer and chooses the cut-line to split a bin exactly in half in order to maintain a regular grid structure. In the presence of large macros, the requirement of a regular bin structure is relaxed. The cut-line of the bin is shifted to allow the largest macro to fit into a child bin after partitioning. If macros can only fit in one bin, they are pre-assigned to the child bin in which they can fit and not involved in partitioning [38, 39].

Bin-based Simulated Annealing after partitioning is also modified as bins may not all have the same dimensions. Horizontal swaps between adjacent bins are only allowed if they are of the same

height. Similarly, vertical swaps between adjacent bins are only allowed if they are of the same width. Lastly, diagonal bin swaps are only legal if the bins have the same height and width. After all bins have a threshold of cells or fewer, partitioning stops and macro locations are legalized. Once legal, macros are considered fixed and partitioning begins again at the top level to place the standard cells of the design [38, 39].

6.2 FengShui

FengShui [4, 27] is a recursive bisection min-cut placer that uses the hMETIS partitioner [26]. FengShui implements the fractional cut technique (see Section 3.2) and packs its placements to either side of the placement region which has a serious affect on the routability of its placements [32]. FengShui also supports mixed-size placement (see Section 4.3)

6.3 NTUPlace2

NTUPlace2 [22] is a hybrid placer that uses both min-cut partitioning and analytical techniques for standard-cell and mixed-size designs. NTUPlace2 uses repartitioning (see Section 3.1), cut-line shifting (see Section 2.3) and weighted net-cut (see Section 3.4) [22].

NTUPlace2 uses analytical techniques to aid partitioning which are different from those in ACG (see Section 3.3). Before partitioning calls to the hMETIS partitioner [26], objects in a placement bin are first placed by an analytical technique to reduce quadratic wirelength [22]. Those objects which are placed far from the proposed cut-line are considered fixed in their current locations for the partitioning process. This technique helps to make terminal propagation more exact and with the weighted net-cut technique has resulted in very good solution quality [22].

To handle mixed-size placement, macro locations are legalized at each layer. Macros become fixed at different layers of placement according to their size relative to placement bin size. Thus larger macros are placed earlier in placement [22]. Macros are legalized using a linear programming technique that attempts to minimize the movement of macros during legalization [22].

6.4 Capo

Capo [30–35] is a min-cut floorplacer. As such it implements the floorplacement flow as described in Section 4.1 and further improved by SCAMPI in Section 5.2 rather than the traditional min-cut flow and implicitly handles mixed-size placement and fixed obstacles in the placement area. Capo

can use either MlPart [10] or hMETIS [26] for hypergraph partitioning. Whitespace allocation in Capo is done per placement bin: either *uniform* (see Section 2.4), *minimum local* or *safe* whitespace allocation (see Section 5.1) is chosen based on the bin's whitespace and user-configurable options. To improve the quality of results, Capo also implements repartitioning (see Section 3.1), placement feedback (see Section 3.1), weighted net-cut (see Section 3.4) and several whitespace allocation techniques. Capo has also been used to optimize Steiner wirelength in placement (see Section 5.3) and can be used for incremental placement (see Section 5.4).

References

- [1] S. N. Adya and I. L. Markov, "Fixed-outline Floorplanning: Enabling Hierarchical Design", *IEEE Trans. on VLSI*, vol. 11, no. 6, pp. 1120-1135, December 2003. (ICCD 2001, pp. 328-334).
- [2] S. N. Adya and I. L. Markov, "Combinatorial Techniques for Mixed-size Placement", *ACM Trans. on Design Autom. of Elec. Sys.*, vol. 10, no. 5, January 2005. (ISPD 2002, pp. 12-17).
- [3] S. N. Adya, I. L. Markov and P. G. Villarrubia, "On Whitespace and Stability in Physical Synthesis," *Integration: the VLSI Journal*, vol. 25, no. 4, pp. 340-362, 2006. (ICCAD 2003, pp. 311-318).
- [4] A. Agnihotri et al., "Fractional Cut: Improved Recursive Bisection Placement," *ICCAD*, pp. 307-310, 2003.
- [5] C. J. Alpert, G.-J. Nam and P. G. Villarrubia, "Effective Free Space Management for Cut-Based Placement via Analytical Constraint Generation," *IEEE Trans. on CAD*, vol. 22, no. 10, pp. 1343-1353, 2003. (ICCAD 2002, pp. 746-751).
- [6] U. Brenner and A. Rohe, "An Effective Congestion Driven Placement Framework," *IEEE Trans. on CAD*, vol. 22, no. 4, pp. 387-394, 2003. (ISPD 2002, pp. 6-11).
- [7] M. Breuer, "Min-cut Placement," *Journal of Design Automation and Fault Tolerant Computing*, vol. 1, no. 4, pp. 343-362, October 1977. (DAC 1977, pp. 284-290).
- [8] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov and A. Zelikovsky, "On Wirelength Estimations for Row-Based Placement", *IEEE Trans. on CAD*, vol. 18, no. 9, pp. 1265-1278, 1999.
- [9] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?," *DAC*, pp. 477-482, Los Angeles, June 2000.
- [10] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Design and Implementation of Move-based Heuristics for VLSI Hypergraph Partitioning," *ACM J. of Experimental Algorithms*, vol. 5, 2000.
- [11] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Optimal Partitioners and End-case Placers for Standard-cell Layout," *IEEE Trans. on CAD*, vol. 19, no. 11, pp. 1304-1314, 2000. (ISPD 1999, pp. 90-96).
- [12] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Hierarchical Whitespace Allocation in Top-down Placement," *IEEE Trans. on CAD* vol. 22, no. 11, pp. 716-724, November 2003.
- [13] Y. C. Chang et al., "B*-trees: A New Representation for Non-Slicing Floorplans," *DAC*, pp. 458-463, 2000.
- [14] T. C. Chen and Y. W. Chang, "Modern Floorplanning Based on Fast Simulated Annealing," *ISPD*, pp. 104-112, 2005.
- [15] T. C. Chen, Y. W. Chang and S. C. Lin, "IMF: Interconnect-Driven Multilevel Floorplanning for Large-Scale Building-Module Designs," *ICCAD*, pp. 159-164, November 2005.
- [16] J. Cong, G. Nataneli, M. Romesis and J. Shinnerl, "An Area-optimality Study of Floorplanning," *ISPD*, pp. 78-83, 2004.
- [17] J. Cong, M. Romesis and J. Shinnerl, "Fast Floorplanning by Look-Ahead Enabled Recursive Bipartitioning," *ASPDAC*, pp. 1119-1122, 2005.
- [18] J. Cong, M. Romesis and J. Shinnerl, "Robust Mixed-Size Placement Under Tight White-Space Constraints," *ICCAD*, pp. 165-172, 2005.
- [19] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard Cell VLSI Circuits," *IEEE Trans. on CAD*, vol. 4, no. 1, pp. 92-98, 1985.
- [20] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions," *DAC*, pp. 175-181, 1982.
- [21] D. J.-H. Huang, and A. B. Kahng, "Partitioning-based Standard-cell Global Placement With an Exact Objective," *ISPD*, pp. 18-25, 1997.

- [22] Z.-W. Jiang et al., "NTUPlace2: A Hybrid Placer Using Partitioning and Analytical Techniques," *ISPD*, pp. 215-217, 2006.
- [23] A. B. Kahng, S. Mantik and I. L. Markov, "Min-max Placement For Large-scale Timing Optimization," *ISPD*, pp. 143-148, April 2002.
- [24] A. B. Kahng, I. I. Mandoiu and A. Zelikovsky, "Highly Scalable Algorithms for Rectilinear and Octilinear Steiner Trees," *ASPDAC*, pp. 827-833, 2003.
- [25] A. B. Kahng and S. Reda, "Placement Feedback: A Concept and Method for Better Min-cut Placement," *DAC*, pp. 357-362, 2004.
- [26] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Domain," *DAC*, pp. 526-629, 1997.
- [27] A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C.-K. Koh and P. H. Madden, "Recursive Bisection Based Mixed Block Placement," *ISPD*, pp. 84-89, 2004.
- [28] C. Li, M. Xie, C. K. Koh, J. Cong and P. H. Madden, "Routability-driven Placement and White Space Allocation," *ICCAD*, pp. 394-401, 2004.
- [29] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter and M. Yildiz, "The ISPD2005 Placement Contest and Benchmark Suite," *ISPD*, pp. 216-220, 2005.
- [30] A. N. Ng, I. Markov, R. Aggarwal and V. Ramachandran, "Solving Hard Instances of Floorplacement," *ISPD*, pp. 170-177, April 2006.
- [31] J. A. Roy, S. N. Adya, D. A. Papa and I. L. Markov, "Min-cut Floorplacement," *IEEE Trans. on CAD*, vol. 25, no. 7, pp. 1313-1326, 2006. (*ICCAD 2004*, pp. 550-557).
- [32] J. A. Roy and I. L. Markov, "Seeing the Forest and the Trees: Steiner Wirelength Optimization in Placement," to appear in *IEEE Trans. on CAD*, 2007. (*ISPD 2006*, pp. 78-85).
- [33] J. A. Roy and I. L. Markov, "ECO-system: Embracing the Change in Placement," to appear in Proceedings of *ASPDAC*, 2007.
- [34] J. A. Roy, D. A. Papa, S. N. Adya, H. H. Chan, J. F. Lu, A. N. Ng and I. L. Markov, "Capo: Robust and Scalable Open-Source Min-cut Floorplacer," *ISPD*, pp. 224-227, April 2005.
- [35] J. A. Roy, D. A. Papa, A. N. Ng and I. L. Markov, "Satisfying Whitespace Requirements in Top-down Placement," *ISPD*, pp. 206-208, April 2006.
- [36] N. Selvakumaran and G. Karypis, "Theto - A Fast, Scalable and High-quality Partitioning Driven Placement Tool," Technical report, Univ. of Minnesota, 2004.
- [37] P. R. Suaris and G. Kedem, "An Algorithm for Quadrisection and Its Application to Standard Cell Placement," *IEEE Trans. on Circuits and Systems*, vol. 35, no. 3, pp. 294-303, 1988. (*ICCAD 1987*, pp. 474-477).
- [38] T. Taghavi, X. Yang, B.-K. Choi, M. Wang and M. Sarrafzadeh "Dragon2005: Large-Scale Mixed-size Placement Tool," *ISPD*, pp. 245-247, April 2005.
- [39] T. Taghavi, X. Yang, B.-K. Choi, M. Wang and M. Sarrafzadeh "Dragon2006: Blockage-Aware Congestion-Controlling Mixed-Size Placer," *ISPD*, pp. 209-211, April 2006.
- [40] K. Takahashi, K. Nakajima, M. Terai, and K. Sato, "Min-cut Placement with Global Objective Functions for Large Scale Sea-of-gates Arrays," *IEEE Trans. on CAD*, vol. 14, no. 4, pp. 434-446, 1995.
- [41] J. Vygen, "Algorithms for Large-Scale Flat Placement", *DAC*, pp. 746-751, 1997.
- [42] E. Wein and J. Benkoski, "Hard macros will revolutionize SoC design", *EE Times*, August 20, 2004. <http://www.eetimes.com/news/design/showArticle.jhtml?articleID=26807055>
- [43] X. Yang, B. K. Choi, and M. Sarrafzadeh, "Routability Driven White Space Allocation for Fixed-die Standard-cell Placement," *IEEE Trans. on CAD*, vol. 22, no. 4, pp. 410-419, April 2003. (*ISPD 2002*, pp. 42-49).
- [44] M. C. Yildiz and P. H. Madden, "Improved Cut Sequences for Partitioning Based Placement," *DAC*, pp. 776-779, 2001.