Module Relocation to Obtain Feasible Constrained Floorplans

Yan Feng and Dinesh P. Mehta, Senior Member, IEEE

Abstract—This paper considers the general problem of relocating modules to convert infeasible constrained floorplanner inputs into feasible ones. This is accomplished by placing modules in locations that attempt to minimize the standard deviation of module densities. Efficient geometric algorithms are developed and shown to be successful in obtaining feasible inputs. Experimental results examine the tradeoffs between achieving a good redistribution of density on one hand and minimizing the increase in wire length and the displacement of modules on the other.

Index Terms—Algorithm, computational geometry, incremental floorplanning, placement, standard deviation.

I. INTRODUCTION

In classical floorplanning, the input consists of a set of (typically rectangular) modules. A set of realizations providing height and width information is associated with each module. In addition, a connectivity matrix that contains the number of interconnections between pairs of modules is provided. A typical objective is to minimize some combination of the area and estimated wire length. A significant body of floorplanning research is concerned with finding a representation that can be used efficiently within the context of simulated annealing [1]–[8].

Kahng [9] critiqued the classical floorplanning problem and proposed a formulation that is more consistent with the needs of current design methodologies. Some of the attributes of his formulation are: 1) the dimensions of the bounding rectangle must be fixed because floorplanning is carried out after the die size and the package have been chosen in most design applications without overlapping in addition to meeting the other requirements set forth by Kahng. The fixed die aspects of Kahng’s formulation have also been addressed by Adya and Markov in their floorplanner tool PARQUET [12]. PARQUET is more suitable for general fixed-outline floorplanning, while our techniques are more suitable for incremental floorplanning.

Mehta and Sherwani [10] developed algorithms that could guarantee zero whitespace whereas Feng et al. [11] developed algorithms that were experimentally shown to obtain zero whitespace. Feng et al. [11] improved on [10] by modifying the problem formulation so that each module is assigned an area within a constraining rectangle. This constraint-based formulation often makes inputs infeasible (see Fig. 1). It is important to be able to detect that an input is not feasible so that unnecessary time and effort are not spent in trying to solve an impossible problem. A max-flow algorithm was used to test the input for feasibility. If the input is feasible, it is possible to actually compute the floorplan. This was accomplished by a min-cost max-flow heuristic. The use of a heuristic was justified because the constrained floorplanning problem (CFP) is NP-hard. However, the flow-based algorithm did not result in connected modules and consequently a postprocessing step was introduced whose principal function was to ensure

Fig. 1. Two constraining rectangles A and B each having an area of 20 units. The area of $A \cap B$ is 4 units. However, the modules that must be contained in A and B both require 19 units. Thus, 38 units of area must be accommodated in a constrained area of 36 units, which is not possible.

Manuscript received July 14, 2004; revised October 7, 2004. This work was supported by the National Science Foundation under Grant CCR-9988338. This paper was recommended by Associate Editor J. Lillis.

The authors are with the Department of Mathematical and Computer Science, Colorado School of Mines, Golden, CO 80401-1887 USA (e-mail: yfeng@mines.edu; dmehta@mines.edu).

Digital Object Identifier 10.1109/TCAD.2005.855981
Density is defined as follows: consider a module requiring that explicitly minimizes the standard deviation of density. This paper is an algorithm that finds a location for a module and moving it to another location in the floorplan where it is desirable properties relating to interconnect such as a good wire length that we do not wish to disrupt.) Our paper addresses the problem by iteratively identifying a badly placed module 

![Diagram](image.png)

**Fig. 2.** Design flow proposed in [11]. The steps contained in the large dashed rectangle are addressed in [11]. This paper proposes an algorithm to solve the problem shown in the solid rectangle.

that modules were connected (i.e., that modules are assigned contiguous area). Recent research [13] indicates that there is often significant whitespace in floorplans. This was exploited in [14], where a cleaner and more effective postprocessing step was introduced to assure that modules are connected in floorplans that contained whitespace. In addition to assuring module connectivity, Feng and Mehta [14] also significantly reduce the number of sides of rectilinear modules. The practical consequence of these two results ([11], [14]) is that bound feasibility virtually guarantees module connectivity. Fig. 2 shows the design flow proposed in [11].

The issue that we address in this paper is how to handle the situation when the input provided to the constrained floorplanner has been shown to be infeasible by the bound-feasibility step. This is a plausible outcome of the preceding step, which is interconnect centric and is oblivious to area considerations. This requires a step (the thick box in Fig. 2) that minimally modifies the input to make it feasible. (Minimally, because even if the input is not feasible with respect to area, it has other desirable properties relating to interconnect such as a good wire length that we do not wish to disrupt.) Our paper addresses the problem by iteratively identifying a badly placed module and moving it to another location in the floorplan where it is unlikely to result in an infeasible layout. A key contribution of this paper is an algorithm that finds a location for a module that explicitly minimizes the standard deviation of density. (Density is defined as follows: consider a module requiring 30 units of area that must be an allotted area within a $5 \times 10$ constraining rectangle. The density contribution of this module is 0.6.) Standard deviation is the traditional statistical metric that calibrates variability of data; it is precisely this variability in density that causes a design to be infeasible. Therefore, we expect that reducing the standard deviation will also make the design feasible. However, solely minimizing the standard deviation might cause a block to be placed very far away from its original location. Therefore, our approach combines standard deviation minimization with proximity considerations.

Relevant details of the CFP are described in Section II. Details of the standard-deviation-based algorithm are provided in Section III. Our experiments (Section IV) demonstrate the viability of our method for 30 benchmarks. Our method was able to convert an infeasible solution into a feasible one on all but one occasion.

### II. REVIEW OF BOUND FEASIBILITY [11]

#### A. Problem Definition

We review the CFP in this section.

Input. 1) An $H \times W$ bounding box that denotes the fixed die. 2) A set of modules $N$ such that each module $m_i \in N$, $1 \leq i \leq n = |N|$, is denoted by a quintuple $(x_i, y_i, w_i, h_i, A_i)$. The coordinates $(x_i, y_i)$ denote the center point of the module and must satisfy $0 \leq x_i \leq W$ and $0 \leq y_i \leq H$. $A_i$ denotes the area units to be assigned to module $m_i$, $w_i$ and $h_i$ denote the module bounds in the $x$ and $y$ directions, respectively.

Output. For each module $m_i$: 1) $A_i$ units of area are assigned to $m_i$; 2) the area assigned to $m_i$ must be inside the fixed die and should be within the constraining rectangle bounded by $[x_i - w_i/2, x_i + w_i/2]$ in the $x$-direction and $[y_i - h_i/2, y_i + h_i/2]$ in the $y$-direction; and 3) $m_i$ must be connected.

Note: A fixed macroblock can be modeled in this formulation by setting $w_i$ and $h_i$ to be its dimensions and by making $A_i = w_i h_i$. If $A_i < w_i h_i$, the problem is not feasible.

CFP was shown to be NP-hard in [11], making it unlikely that a fast algorithm exists to solve CFP.

#### B. Feasibility

Let $Q = \{Q_i | 1 \leq i \leq n\}$ be the set of $n$ constraining rectangles, each of which corresponds to a module.

**Definition:** An input to CFP is said to be bound feasible if for any subset $T \in N$ (i.e., the set of all modules) we have

$$\sum_{m_i \in T} A_i \leq \text{Area} (\bigcup_{m_i \in T} Q_i). \quad (1)$$

The constraining rectangles disect the plane into $m$ regions, where a region is defined as a connected set of all points that belong to the same subset of $Q$. Fig. 3 demonstrates how the bounding rectangle is decomposed into regions by a set of constraining rectangles. The constraining rectangles and regions are used to define a flow network that is used to determine bound feasibility.
III. THE STANDARD DEVIATION MINIMIZATION ALGORITHM

This section discusses the main technical contribution of this paper.

A. Theory

Each module $m_j \in N$ has a module density $\omega_j$, which is the fraction of its constraining rectangle’s area that it needs, and is given by $A_i/(w_j \times h_j)$. This quantity is 1 for a hard macro. As mentioned before, the constraining rectangles dissect the plane into $m$ regions. Consider a region $r_i$ that is the intersection of constraining rectangles $Q_{i_1}, Q_{i_2}, \ldots, Q_{i_k}$. Then, the region density for $r_i$ is given by $\Omega_i = \sum_{j=1}^{k} \omega_j$. The overall average region density is $\mu = \sum_{i=1}^{m} \Omega_i \times \text{Area}(r_i)/A$, where $A$ is the area of fixed die. Let $W = ((A - \sum_{i=1}^{n} A_i)/A)$ be the fraction of the floorplan that contains whitespace.

**Theorem 1:** $\mu = 1 - W$.

**Proof:**

\[
\mu = \frac{\sum_{i=1}^{m} \Omega_i \times \text{Area}(r_i)}{A} \\
= \frac{\sum_{j=1}^{n} \omega_j \times (w_j \times h_j)}{A} \\
= \frac{\sum_{j=1}^{n} A_i}{A} \\
= 1 - W.
\]

For a given input, $W$ is constant, so $\mu$ is constant; i.e., the average region density is a constant no matter where you put the modules. In order to make it likely that an input is bound feasible, we would like the region density to be evenly distributed. This can be accomplished by reducing the standard deviation of the region’s density, which is defined as

\[
\sigma = \sqrt{\frac{1}{A} \sum_{i=1}^{m} a_i (\Omega_i - \mu)^2}.
\]

Here, $a_i$ denotes the area of region $r_i$. A smaller $\sigma$ denotes a better input with evenly distributed region density across the floorplan. Thus, our strategy will be to identify a module that is causing the input to be bound infeasible and move its constraining rectangle to a part of the floorplan that minimizes standard deviation. Although there are potentially an infinite number of locations in which to place a constraining rectangle to obtain minimum $\sigma$, we show below that our search for an optimal location can be restricted to a finite set of choices.

**Theorem 2:** Consider a module $R$ that is being added to an input consisting of $n$ constraining rectangles. The number of locations for $R$ that must be examined for minimum $\sigma$ is at most $16n^2$.

**Proof:** The claim is proved here is that there exists a location for $R$ with minimum $\sigma$ such that one horizontal side of $R$ coincides with one of the $2n$ horizontal sides of the $n$ constraining rectangles in the input. Since $R$ has two horizontal sides, there are $4n$ possibilities. A symmetric argument shows that there are $4n$ ways for the vertical sides to coincide. Combining the two statements results in a maximum of $16n^2$ possibilities.

It remains to show that there exists a minimum $\sigma$ location for $R$ so that one of $R$’s horizontal sides coincides with an existing horizontal side. We accomplish this by starting with a location for $R$ with minimum $\sigma$ that has no coinciding sides and then by “sliding” $R$ to a location that does have coinciding sides without changing $\sigma$.

Assume that a constraining rectangle $R$ of height $h$ and width $w$ is placed in a location with minimum $\sigma$. A small rectangle $R_{\text{top}}$ of height $\epsilon$ and width $w$ is constructed at the top portion of $R$ as shown in Fig. 4(a). The quantity $\epsilon$ is chosen so that neither rectangle $R_{\text{top}}$ nor the rectangle of height $\epsilon$ and width $w$ [the shaded region in Fig. 5(b)] below $R$ intersects any existing horizontal edges. We assume that $t - 1$ vertical edges intersect $R_{\text{top}}$, partitioning it into $t$ subrectangles. A rectangle $R_{\text{bottom}}$ of height $\epsilon$ can be similarly constructed in the lower portion of $R$. Assume that $R_{\text{bottom}}$ can be partitioned into $b$ subrectangles by vertical edges. Then, both $R_{\text{top}}$ and $R_{\text{bottom}}$ can be partitioned into at most $t + b - 1$ rectangles so that each corresponding rectangle (one from $R_{\text{top}}$ and the one from
The positions of module $R$ are denoted by $\alpha_i$s and $\beta_i$s, respectively. 

Observe that if we move rectangle $R$ [Fig. 5(a)] down by a small distance $\epsilon$, only densities within the shaded region [shown in Fig. 5(b)] will change; i.e., in order to compare $\sigma$ before and after movement, only the shaded areas need to be taken into account.

The following equations calculate $A\sigma^2$ before ($A\sigma_{\text{old}}^2$) and after ($A\sigma_{\text{new}}^2$) the move, respectively. $a_i$ denotes the area of each small rectangle, $\mu$ is the mean density for the entire floorplan, and $z$ is the density of module $R$. $C$ is the constant part of the standard deviation that is not affected by the movement. We obtain the equations

\[
A\sigma_{\text{old}}^2 = \sum_{i=1}^{t+b-1} a_i(\alpha_i - \mu)^2 + \sum_{i=1}^{t+b-1} a_i(\beta_i - \mu - z)^2 + C
\]

\[= \sum_{i=1}^{t+b-1} a_i(\alpha_i - \mu)^2 + \sum_{i=1}^{t+b-1} a_i(\beta_i - \mu)^2 - 2 \sum_{i=1}^{t+b-1} a_i(\beta_i - \mu)z + \sum_{i=1}^{t+b-1} a_i z^2 + C\]

\[
A\sigma_{\text{new}}^2 = \sum_{i=1}^{t+b-1} a_i(\alpha_i - \mu - z)^2 + \sum_{i=1}^{t+b-1} a_i(\beta_i - \mu)^2 + C
\]

\[= \sum_{i=1}^{t+b-1} a_i(\alpha_i - \mu)^2 - 2 \sum_{i=1}^{t+b-1} a_i(\alpha_i - \mu)z + \sum_{i=1}^{t+b-1} a_i(\beta_i - \mu)^2 + \sum_{i=1}^{t+b-1} a_i z^2 + C.\]

Since $\sigma_{\text{old}}$ was assumed to be minimum, $\sigma_{\text{old}} \leq \sigma_{\text{new}}$, giving

\[
A\sigma_{\text{old}}^2 \leq A\sigma_{\text{new}}^2
\]

\[
-2 \sum_{i=1}^{t+b-1} a_i(\beta_i - \mu)z \leq -2 \sum_{i=1}^{t+b-1} a_i(\alpha_i - \mu)z + \sum_{i=1}^{t+b-1} a_i(\alpha_i - \mu)z + \sum_{i=1}^{t+b-1} a_i \beta_i. \quad (2)
\]

From (2) and (3), we have

\[
\sum_{i=1}^{t+b-1} a_i \alpha_i = \sum_{i=1}^{t+b-1} a_i \beta_i
\]

making $\sigma_{\text{old}} = \sigma_{\text{new}}$, which means we can move module $R$ upward (downward) until it hits another horizontal edge without affecting the standard deviation; i.e., there exists a location of $R$ with minimum $\sigma$ such that a horizontal edge of $R$ coincides with a horizontal edge of a rectangle in the input.

**B. Algorithm**

Our algorithm exploits the observation of Theorem 2 by attempting to place the bounding rectangle $R$ at each of the up to $16n^2$ locations and evaluating $\sigma$ each time. In order to accomplish this, our algorithm must 1) identify the up to $16n^2$ locations for $R$ and 2) compute region densities $\Omega_i$ for each region. An outline of the algorithm follows.

1. **Algorithm** find_min_sd($R$)
2. **begin**
3. // Step I. Get region densities to the right of each vert. edge.
4. Sort the vertical edges of rectangles from left to right
5. **For each** $x$-position in the sorted list of edges
6. Update the region densities to the right of the edge
7. // Step II. Find set $S = \{s_1, s_2, \ldots, s_{2n}\}$, where $s_i$
8. // is the position with minimal $\sigma$ at some horizontal edge.
In each horizontal edge in the floorplan, an ordered list of horizontal edges is already in the layout. Note that a similar computation would need to be carried out when the virtual strip is below $E_i$. Step I computes region density information by sweeping a horizontal line across the plane from bottom to top; the event points are horizontal edges of rectangles that intersect with the virtual strip. There are three possible positions for $R$ (the shaded rectangles) such that its lower horizontal edge coincides with $E_i$ and one of its vertical edges with one of the vertical edges already in the layout. Note that a similar computation would need to be carried out when the virtual strip is below $E_i$.

Fig. 7. $E$ is a horizontal edge of a rectangle in the floorplan. A horizontal side of $R$ can coincide with $E$ in the two ways shown.

Fig. 8. Thick horizontal line represents the current horizontal edge $E_i$. The dashed line box is the virtual strip. The two thick vertical lines represent vertical edges of rectangles that intersect with the virtual strip. There are three possible positions for $R$ (the shaded rectangles) such that its lower horizontal edge coincides with $E_i$, and one of its vertical edges with one of the vertical edges already in the layout. Note that a similar computation would need to be carried out when the virtual strip is below $E_i$.

C. General Case of Implicit Comparison

So far, we have discussed how to examine the up to $16n^2$ positions to find a position with minimum $\sigma$. Explicitly computing $\sigma$ at each possible location takes $\Theta(m)$ time, where $m$, the number of regions in the entire floorplan, is $O(n^2)$. This is expensive. Here, we will introduce an implicit comparison algorithm that can be used to identify the best location without explicitly computing $\sigma$. This may be used in Line 16 of Algorithm `find_min_sd` to identify the minimum $\sigma$ location.

Let $P_1$ and $P_2$ be two possible locations for rectangle $R$. In order to determine which position has smaller $\sigma$, we first place $R$ at position $P_1$, where $R$ is dissected into $k$ regions with area $A_i$ and density $\alpha_i$ as shown in Fig. 10(a). Then $R$ is placed at position $P_2$, where $R$ is dissected into $l$ regions with area $B_i$ and density $\beta_i$ as shown in Fig. 10(b).

Equation (5), which is derived below, shows how to compare $\sigma$ between $P_1$ and $P_2$, i.e.,

\[
x A \sigma_{P_1}^2 = \sum_{i=1}^{k} A_i (\alpha_i - \mu)^2 + \sum_{i=1}^{k} B_i (\beta_i - \mu - z)^2 + C
\]

\[
A \sigma_{P_2}^2 = \sum_{i=1}^{k} A_i (\alpha_i - \mu - z)^2 + \sum_{i=1}^{k} B_i (\beta_i - \mu)^2 + C
\]

\[
A \left( \sigma_{P_1}^2 - \sigma_{P_2}^2 \right) = -2z \sum_{i=1}^{k} B_i \beta_i + 2z \mu \sum_{i=1}^{k} B_i + z^2 \sum_{i=1}^{k} B_i
\]

\[
+ 2z \sum_{i=1}^{k} B_i \beta_i - 2z \mu \sum_{i=1}^{k} B_i - z^2 \sum_{i=1}^{k} B_i
\]

(4)

Since

\[
A \sum_{i=1}^{k} A_i = \sum_{i=1}^{l} B_i
\]

(4) becomes

\[
A \left( \sigma_{P_1}^2 - \sigma_{P_2}^2 \right) = 2z \left( \sum_{i=1}^{k} A_i \alpha_i - \sum_{i=1}^{l} B_i \beta_i \right)
\]

(5)

If this quantity is positive, location $P_2$ has a smaller $\sigma$ than location $P_1$; otherwise, $P_1$ has a smaller $\sigma$ than $P_2$. This test may be used to compare the two positions in $\Theta(k + l)$ time rather than $\Theta(m)$ time. This will improve the complexity as described in Section III-E.

D. Implicit Comparison at Each Horizontal Edge

Equation (5) can be further simplified if we only want to compare $\sigma$ between consecutive positions within the same virtual strip. This may be used to implement the comparisons in Line 14 of Algorithm `find_min_sd`.

There are two cases. First, suppose the two consecutive positions $P_1$ and $P_2$ are not overlapped as shown in Fig. 11. Then
Fig. 9. (a) Initial input consisting of four rectangles (excluding \( R \)) in a 30 × 30 bounding rectangle. Assume \( R \) has both width and height equal to 10. All module densities are assumed to be 0.8. (b) Result of Step I that obtains the density to the right of each vertical edge. The asterisks in (c) show all possible positions \( P_i \) for the lower left point of module \( R \) for the first event point in the plane sweep algorithm of Step II—in the example, the first event point is the horizontal edge corresponding to the lower boundary of the fixed die. (d) Asterisks to show three possible positions \( (s_i) \), each corresponding to a different horizontal edge. Step III of the algorithm chooses the best among these positions.

Fig. 10. Implicit comparison between two possible positions \( P_1 \) and \( P_2 \). \( P_1 \) \((P_2)\) is dissected into three different regions. (a) Module at position \( P_1 \). Both figures include region density and area information.

Then (5) becomes

\[
A \left( \sigma_{P_1}^2 - \sigma_{P_2}^2 \right) = 2wz \left( \sum_{i=1}^{k} d_i \alpha_i - \sum_{i=1}^{l} h_i \beta_i \right) \quad (6)
\]

where \( w \) is the width of each rectangle in Fig. 11. Second, suppose the two consecutive positions \( P_1 \) and \( P_2 \) overlap with each other as shown in Fig. 12.

The density of the overlapped region \( O \) in Fig. 12(a) is the same for each position. So the difference of \( \sigma \) at two positions is only determined by the difference between the two shaded rectangles as shown in Fig. 12(b). Also, no vertical
Fig. 12. Overlapped consecutive positions. (a) Two overlapped rectangles and the rectangle $O$ in the middle is the overlapped area. (b) In order to compare the difference of $\sigma$, we only need to know the difference between two shaded rectangles.

edge intersects the two shaded boxes. Otherwise, these two are not consecutive positions. Similar to the first case, each shaded rectangle is dissected into several rectangles with the same width. Assume $\alpha_i$, $\beta_i$, $d_i$, and $h_i$ are defined as in the nonoverlapping case. We obtain the same equation as (6). In general, in order to compare two consecutive positions from same horizontal edge, only region density information to the right of the left and right edge of each position is needed as shown in (6). This will improve the complexity as described below.

E. Data Structure and Time Complexity

A naive implementation of Algorithm find_min_sd requires $O(n^4)$ time. Computing $\sigma$ requires $O(n^2)$ time. From Theorem 2, the total number of $\sigma$ computations is $O(n^2)$. The implicit methods described in the preceding sections for comparing $\sigma$s are designed to improve this. Further improvement may be obtained by using geometric data structures such as the 2-D range tree and 2-D interval trees to identify all possible locations of $R$. These data structures and their application of identifying all the edges that intersect a rectangle are discussed in detail in [15].

In Step I, we construct a 2-D range tree 2DRT and a 2-D interval tree 2DIT of all vertical edges. These two trees are needed in Steps II and III. Also, a balanced binary search tree BST$_i$ is constructed at each vertical edge to record the region density information to the right of the edge. (This sequence of BSTs implements line 6 of Algorithm find_min_sd.) The time complexity for construction of 2DRT and 2DIT is $O(n \log n)$. The time complexity for construction of BST$_i$ is $O(n \log n)$.

Next, we compute BST$_i$, $i > 1$. Assume BST$_{i-1}$ is available and $e$ is the next vertical edge in left-to-right order. To construct BST$_i$, we first copy BST$_{i-1}$, which takes $O(n)$ time. Then we update the intervals that overlap with $e_i$. Fig. 13 illustrates the steps when the left edge $e$ of a rectangle is encountered. A similar set of steps is needed when the right edge of a rectangle is encountered.

From Fig. 13(b) and (c), updating the interval that contains one endpoint of $e$ needs one deletion and two insertion operations. For both endpoints of $e$, we require six operations. Each insertion/deletion requires $O(\log n)$ time. Updating the densities of the intervals contained in $e$ requires $O(j)$ time, where $j$ is the number of intervals that needs to be updated. Thus, the time required to update BST$_i$ because of edge $e$ is $O(\log n + j) = O(n)$. A similar bound holds for edge deletion. BST$_i$ can thus be computed from BST$_{i-1}$ in $O(n)$ time. The overall time complexity of Step I is bounded by $O(n^2)$.

Fig. 13. (a) Intervals from BST$_{i-1}$, their densities, and a new vertical edge $e$ belonging to a rectangle with density $z$. (b) Deletion of the interval that contains one endpoint of $e$. (c) Insertion of two new intervals and their densities. (d) Same procedure for the other endpoint of $e$ and also the updated densities of all intervals that are contained in $e$.

From Fig. 13(b) and (c), updating the interval that contains one endpoint of $e$ needs one deletion and two insertion operations. For both endpoints of $e$, we require six operations. Each insertion/deletion requires $O(\log n)$ time. Updating the densities of the intervals contained in $e$ requires $O(j)$ time, where $j$ is the number of intervals that needs to be updated. Thus, the time required to update BST$_i$ because of edge $e$ is $O(\log n + j) = O(n)$. A similar bound holds for edge deletion. BST$_i$ can thus be computed from BST$_{i-1}$ in $O(n)$ time. The overall time complexity of Step I is bounded by $O(n^2)$.

Lines 11–13 of Step II in Algorithm find_min_sd are implemented by using 2DRT and 2DIT to find all the vertical edges that intersect the virtual strip corresponding to each horizontal edge. This takes $O(\log^2 n + k_v)$, where $k_v$ is the maximum number of vertical edges that intersects with any horizontal strip [15]. The right-hand side of (6) is used to carry out the implicit comparison of $\sigma$ between two consecutive positions (Line 14 of Algorithm find_min_sd). The information needed to
TABLE I

RESULTS FOR ami49 MODIFIED USING THE “HIGHLY INFEASIBLE” STRATEGY. OF THE 15 EXPERIMENTS PERFORMED, ONLY ONE INPUT REMAINED INFEASIBLE. IN ALMOST ALL CASES, ACHIEVING FEASIBILITY WAS ACCOMPANIED BY AN INCREASED WIRE LENGTH.
THE TOP TEN METHOD YIELDS BETTER DISPLACEMENT VALUES IN 13 CASES AND A BETTER WIRE LENGTH IN NINE CASES RELATIVE TO THE MINIMUM \( \sigma \) METHOD. FOR A GIVEN \( \gamma \), THE WIRE LENGTH USUALLY INCREASES AS THE WHITESPACE INCREASES, BUT THE DISTANCE BY WHICH MODULES HAVE TO BE MOVED TO ACHIEVE FEASIBILITY DECREASES.

<table>
<thead>
<tr>
<th>( \gamma )</th>
<th>white space</th>
<th>Initial HPWL</th>
<th>minimum ( \sigma )</th>
<th>top ten ( \sigma )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>HPWL</td>
<td>Displacement</td>
<td>% blocks moved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>total</td>
<td>average</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>10%</td>
<td>889440</td>
<td>881241</td>
<td>10255</td>
</tr>
<tr>
<td></td>
<td>15%</td>
<td>889153</td>
<td>909638</td>
<td>9564</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>923991</td>
<td>950060</td>
<td>3460</td>
</tr>
<tr>
<td>0.75</td>
<td>10%</td>
<td>885963</td>
<td>881241</td>
<td>11194</td>
</tr>
<tr>
<td></td>
<td>15%</td>
<td>888792</td>
<td>896195</td>
<td>7978</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>913368</td>
<td>936286</td>
<td>4961</td>
</tr>
<tr>
<td>0.8</td>
<td>10%</td>
<td>861939</td>
<td>881377</td>
<td>15556</td>
</tr>
<tr>
<td></td>
<td>15%</td>
<td>884654</td>
<td>901291</td>
<td>2533</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>909139</td>
<td>920156</td>
<td>8432</td>
</tr>
<tr>
<td>0.85</td>
<td>10%</td>
<td>848411</td>
<td>889822</td>
<td>22981</td>
</tr>
<tr>
<td></td>
<td>15%</td>
<td>870796</td>
<td>880312</td>
<td>17930</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>894961</td>
<td>902539</td>
<td>12224</td>
</tr>
<tr>
<td>0.9</td>
<td>10%</td>
<td>850708</td>
<td>inffeasible</td>
<td>inffeasible</td>
</tr>
<tr>
<td></td>
<td>15%</td>
<td>830450</td>
<td>888077</td>
<td>32475</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>853692</td>
<td>908712</td>
<td>32186</td>
</tr>
</tbody>
</table>

compute the right-hand side of the equations (namely, densities and heights) is obtained in \( O(\log n + k_h) \) time from the appropriate BSTs, where \( k_h \) is the maximum number of intervals reported at any BST. Since there are \( O(k_v) \) positions, the total time complexity is \( O(k_v(k_h + \log n) + \log^2 n) \). The overall time complexity of Step II is \( O(nk_v(k_h + \log n) + n \log^2 n) \) because there are \( O(n) \) virtual strips.

Line 16 of Step III finds the position with minimal \( \sigma \) among candidates from each horizontal edge. Based on the discussion in Section III-C and (5), to determine the difference between \( \sigma \) from two arbitrary positions, we have to find the vertical edges that intersect with the module that takes \( O(\log^2 n + k_v) \) time using 2DRT and 2DIT. Also, the region density information at each vertical edge is required, which may be computed in \( O(\log n + k_h) \) time from the appropriate BSTs. Because there are \( O(k_v) \) vertical edges, the time complexity to compare \( \sigma \) between two arbitrary positions is \( O(k_v(k_h + \log n) + \log^2 n) \). The overall time complexity of Step III is \( O(nk_v(k_h + \log n) + n \log^2 n) \) because there are \( O(n) \) horizontal edges.

In the worst case, \( k_v \) and \( k_h \) are \( O(n) \). Therefore, the overall worst case time complexity is \( O(n^3) \). Only pathological cases where every vertical edge intersects every horizontal edge gives these results. In practice, we expect \( k_v \) and \( k_h \) to be constant, so the running time of our algorithm is \( O(n^2) \). Our preliminary experiments with large floorplans (300 modules) show that our run times are on the order of a few seconds.

F. Infeasible Input Conversion

The preceding sections describe how only one rectangle is relocated so as to minimize \( \sigma \). In order to convert a bound-infeasible input to a bound-feasible one, several rectangles may have to be moved. The following describes the complete process of how an infeasible input is converted into a feasible one.

Algorithm Make_Feasible

while infeasible // use max flow [11] to determine feasibility
Use the results of the max-flow algorithm to identify the module \( R \) with the most deficient area = required area – actual assigned area.
if \( (R) \) is the same as the previous iteration
Select a rectangle that intersects \( R \) and call it \( R \).
Remove \( R \) from input
find_min_sd(R)
Remove \( R \) from input
Update input
end while

The previous algorithm does not guarantee that a feasible input will be obtained. (However, our experimental results show that it does so in almost every test case when the percentage of whitespace is at least 10%.) A more serious problem is that \( \text{find_min_sd}(R) \) might move \( R \) by a large distance. Assuming that the initial input was obtained in an effort to optimize wire length, this redistribution could result in a significant degradation of wire length. Independent of the wire length, a large movement of modules could negate the gains of the preceding...
TABLE II
RESULTS FOR ami49 MODIFIED USING THE "ALMOST FEASIBLE" STRATEGY. FEASIBILITY WAS ACHIEVED IN EVERY CASE.

UNLIKE IN THE PREVIOUS TABLE, WIRE LENGTH WAS NOT SIGNIFICANTLY AFFECTED AS A RESULT OF MAKING THE INPUT FEASIBLE. THIS IS TO BE EXPECTED AS THE INPUT IS "ALMOST FEASIBLE". THE TOP TEN METHOD OUTPERFORMED THE MINIMUM \( \sigma \) METHOD WITH RESPECT TO WIRE LENGTH; HOWEVER, A CLEAR WINNER DID NOT EMERGE WITH RESPECT TO WIRE LENGTH. AN INCREASE IN WHITESPACE WITH CONSTANT \( \gamma \) RESULTS IN INCREASED WIRE LENGTHS, BUT SMALLER DISPLACEMENT.

<table>
<thead>
<tr>
<th>( \gamma )</th>
<th>white space</th>
<th>Initial HPWL</th>
<th>minimum ( \sigma )</th>
<th>top ten ( \sigma )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>HPWL Displacement</td>
<td>% blocks moved</td>
<td>HPWL Displacement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total Average</td>
<td></td>
<td>Total Average</td>
</tr>
<tr>
<td>0.2</td>
<td>10%</td>
<td>878037</td>
<td>896239</td>
<td>2390 48.8</td>
</tr>
<tr>
<td></td>
<td>15%</td>
<td>898557</td>
<td>892922</td>
<td>995 20.3</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>922560</td>
<td>926590</td>
<td>200 4.08</td>
</tr>
<tr>
<td>0.3</td>
<td>10%</td>
<td>880175</td>
<td>863432</td>
<td>7311 149.2</td>
</tr>
<tr>
<td></td>
<td>15%</td>
<td>902940</td>
<td>924869</td>
<td>2442 49.8</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>933972</td>
<td>931613</td>
<td>818 16.7</td>
</tr>
<tr>
<td>0.4</td>
<td>10%</td>
<td>885864</td>
<td>888812</td>
<td>4213 86.0</td>
</tr>
<tr>
<td></td>
<td>15%</td>
<td>894380</td>
<td>895054</td>
<td>1538 31.4</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>930549</td>
<td>916319</td>
<td>2576 52.6</td>
</tr>
<tr>
<td>0.5</td>
<td>10%</td>
<td>860080</td>
<td>848922</td>
<td>14628 298.5</td>
</tr>
<tr>
<td></td>
<td>15%</td>
<td>918977</td>
<td>921700</td>
<td>1625 33.1</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>931088</td>
<td>930875</td>
<td>848 17.3</td>
</tr>
</tbody>
</table>

Fig. 16. Changes in density distribution on an "almost-feasible" benchmark on ami33 with 15% whitespace.

IV. EXPERIMENTAL RESULTS

Code was developed in C++ for a Linux workstation running on a Pentium III processor. We derived benchmarks from MCNC benchmarks ami33 and ami49 as follows: the initial floorplans, which were obtained from the VLSI CAD Bookshelf, are modified by using a parameter \( W \) that represents the desired percentage of whitespace. The size of the fixed die was modified to obtain the desired level of whitespace. Suppose that the total required area is \( A \) and the die size of the initial floorplan is \( W \times H \). To get 20% whitespace, we scaled both \( W \) and \( H \) by a factor \( \alpha \) such that \( (\alpha \times W)(\alpha \times H) = (1 + 0.2)A \). The following three subsections examine the performance of our algorithms on three different types of benchmarks. We used LEDA [16] to perform the feasibility computation described in Section II. We have only shown our results on ami49 as the results on ami33 followed similar trends. Section IV-C uses our algorithms to remove overlaps between macroblocks on the output of the force-directed macroblock placer in [17], while the last subsection discusses scalability issues.

A. "Highly Infeasible" Benchmark

This benchmark is constructed by using a parameter \( \gamma \) that is a percentage value. Suppose \( \gamma \) is 80% for a given benchmark. Then, all modules that are in the left 80% of the fixed die area are moved left so that they occupy the left half (50%) of the die. This usually results in two clusters of modules (one on the left and one on the right) separated by a strip of whitespace. This was chosen to create situations where the input is highly infeasible and to see whether standard deviation minimization can make the input feasible. The larger the \( \gamma \), the more infeasible the benchmark. Table I shows the results of using the two standard deviation algorithms presented in the previous section. HPWL refers to the half-perimeter wire length. Total Displacement is the sum of the Manhattan distances by which rectangles had to be displaced.

Fig. 15 shows the impact of our algorithms on density distribution.

B. "Almost-Feasible" Benchmark

This benchmark is constructed by using a parameter \( \gamma \) that is a percentage value. We randomly select \( \gamma \) percent of all modules and move each module in one of four directions (up, left, right, down) chosen randomly by 100 units. This was chosen to create situations where the input is almost feasible and to see whether standard deviation minimization can go the "last mile" to make the input feasible. Table II shows the results of our experiments. Fig. 16 shows the impact of our algorithms on density distribution.
TABLE III
RESULTS FOR ami49 MODIFIED USING THE "FORCED WHITESPACE BENCHMARK." IN TWO CASES, BOTH WIRE LENGTH AND DISPLACEMENT IMPROVED AS A RESULT OF USING THE "TOP TEN" VERSION OF THE STANDARD DEVIATION MINIMIZATION ALGORITHM RELATIVE TO THE ORIGINAL STANDARD DEVIATION MINIMIZATION ALGORITHM. IN THE THIRD CASE, BOTH ALGORITHMS OBTAINED IDENTICAL SOLUTIONS.

<table>
<thead>
<tr>
<th>whitespace</th>
<th>Initial HPWL</th>
<th>Initial Displacement</th>
<th>minimum σ</th>
<th>% blocks moved</th>
<th>top ten σ</th>
<th>% blocks moved</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>876901</td>
<td>882677</td>
<td>3829</td>
<td>78.1</td>
<td>14.2</td>
<td>875276</td>
</tr>
<tr>
<td>15%</td>
<td>904139</td>
<td>924658</td>
<td>1147</td>
<td>23.4</td>
<td>10.2</td>
<td>902833</td>
</tr>
<tr>
<td>20%</td>
<td>910391</td>
<td>913483</td>
<td>971</td>
<td>19.8</td>
<td>10.2</td>
<td>913483</td>
</tr>
</tbody>
</table>

Fig. 17. Changes in density distribution on an “forced whitespace” benchmark on ami33 with 15% whitespace.

C. Forced Whitespace Benchmark

We randomly create some 200 × 200 areas and move any constraining rectangles that intersect these areas so that they no longer intersect these areas. The idea behind this benchmark is to force the creation of predefined whitespace areas. Table III shows the results of our experiments.

Fig. 17 shows the impact of our algorithms on density distribution.

D. Overlap Removal

Next, we decided to stress our algorithm by applying it to a scenario for which it was not designed; namely, removing overlap from a floorplan that consists solely of macroblocks without increasing the die size. This is modeled by assigning each block a density of one and by modifying the feasibility test so that a density greater than one anywhere in the floorplan (signifying the overlap of two modules) renders the floorplan infeasible. Our algorithms are used to remove overlaps between macroblocks on the output of the force-directed macroblock placer in [17]. The force-directed algorithm attempts to iteratively improve the placement and remove overlaps. However, intermediate iterations still have several overlaps. We selected the results at the end of four of these iterations to illustrate our algorithms.

Based on our experience with the benchmarks previously presented, we modified the “top ten” method as follows. First, we use the minimum σ method to determine the location of the module. If the distance by which the module is moved is within a certain range (we used 100 units), we use the new location. If the distance is out of range, we use the “top ten” method. Table IV shows the results of our experiments.

We also experimentally compared our overlap removal method with that in [17] as follows. Our starting point was the locations of the blocks at an intermediate iteration in the force-directed solver. The force-directed solver was then run up to the end. However, this did not eliminate all of the overlaps. On the suggestion of one of the authors in [17], we used a radial shifting procedure to eliminate the overlaps. We used the resulting die size to test our overlap removal algorithm. In every case, our algorithm was able to remove overlaps in the prescribed die area. Table V compares the wire lengths of both approaches. Our approach requires less wire length in four out of five cases.

E. Scalability

The average run time for one iteration of Algorithm Make_Feasible (i.e., the process of moving a single rectangle to its final location) was 0.986 s for ami33 and 2.749 s for ami49. This includes the time to run the max-flow-based algorithm that determines feasibility. If this is excluded, these quantities are 0.11 and 0.47 s, respectively. To determine scalability, we ran our algorithm on the GSRC n300 benchmark, which contains 300 modules. The run time for a single call to Algorithm find_min_sd computation is 2.68 s.
V. CONCLUSION

An iterative method consisting of moving modules to improve standard deviation was proposed to convert an unevenly dense distribution of modules to a more even distribution. Although this approach was developed in the context of a design flow for mixed designs, it also had a fair amount of success in the much more difficult context of designs consisting solely of hard macros. An important outcome of the experimental results is that iterative improvement of standard deviation does indeed result in feasible designs. One of the keys to the success of this approach is the availability of whitespace in the design, which is a reasonable assumption in modern designs. In particular, Caldwell et al. [13] state that anecdotal evidence suggests that whitespace varies from 20% to 70%. Our experiments varied whitespace from 10% to 30% and confirmed the intuition that the likelihood of converting infeasible designs into feasible ones is greater when there is more whitespace. Our results also confirm the intuition that the greater the whitespace, the more the wire length and the less modules have to be moved. A less tangible contribution of this paper is based on the observation that there are several parameters that are used to measure the quality of a physical layout such as routing congestion and thermal hotspots, etc., where one wants to evenly distribute the parameter throughout the floorplan. Although this paper addresses this problem for density, we believe that there may be other opportunities to utilize standard deviation minimization techniques similar to the one presented here for these other parameters.

REFERENCES


Dinesh P. Mehta (S’90–M’92–SM’04) received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Bombay, India, in 1987, the M.S. degree in computer science from the University of Minnesota, Minneapolis, in 1990, and the Ph.D. degree in computer science from the University of Florida, Gainesville, in 1992.

In 1996 and 1997, he was a Visiting Professor at Intel’s Strategic CAD Labs. From 1992 to 2000, he was with the University of Tennessee Space Institute. Since 2000, he has been with the Mathematical and Computer Sciences Department, Colorado School of Mines, Golden, where he is currently Professor and Assistant Department Head. He is the coauthor of Fundamentals of Data Structures in C++ and coeditor of Handbook on Data Structures and Applications. He has published 30 journal and conference papers, several of which appear in IEEE and ACM Transactions. His research interests are in very large scale integration (VLSI) design automation, and applied algorithms and data structures.

Dr. Mehta is currently an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: REGULAR PAPERS. He received the Vice President’s Award for Teaching Excellence from the University of Tennessee Space Institute in 1997.

Yan Feng received the B.S. degree in engineering from the University of Science and Technology, Beijing, China, in 1995, and the M.S. degree and the Ph.D. degree in computer science from the Colorado School of Mines, Golden, in 2001 and 2004, respectively.

He is currently a Post-Doctoral Researcher at the Electrical and Computer Engineering Department, University of Minnesota, Minneapolis. His research interests include design and analysis of algorithms and computer-aided design (CAD) of very large scale integration (VLSI) physical design.

Received 2006.