VII. Conclusion

We present a simultaneous buffer insertion/sizing and wire sizing zero skew clock-tree optimization algorithm, *ClockTune*. The algorithm takes polynomial runtime and memory usage and finds minimum-delay and minimum-power embeddings efficiently. For wire widths from 0.3 to 3 μm and buffer widths from $1\times$ to $10\times$, the algorithm achieves $45\times$ delay improvement and $1.25\times$ power-saving over $r5$'s initial routing with 1 μm wires generated by the BB+DME algorithm. *ClockTune* can also be applied for clock tuning to speedup design convergence.

References

[1] ClockTune [Online]. Available: http://vlsi.ece.wisc.edu/Tools.htm

[2] J. G. Xi and W. W.-M Dai, "Useful-skew clock routing with gate sizing for low power design," in *Proc. 33rd Annu. Design Automation Conf.*, 1996, pp. 383–388.

[3] J. Cong, Z. Pan, L. He, C.-K. Koh, and K.-Y. Khoo, "Interconnect design for deep submicron ICs," in *Proc. 1997 IEEE/ACM Int. Conf. Computer-Aided Design*, 1997, pp. 478–485.

[4] X. Zeng, D. Zhou, and W. Li, "Buffer insertion for clock delay and skew minimization," in *Proc. 1999 Int. Symp. Physical Design*, 1999, pp. 36–41.

[5] S. S. Sapatnekar, "RC interconnect optimization under the elmore delay model," in *Proc. 31st Annu. Design Automation Conf.*, 1994, pp. 387–391.

[6] R. Kay, G. Bucheuv, and L. T. Pileggi, "EWA: Exact wiring-sizing algorithm," in *Proc. Int. Symp. Physical Design*, 1997, pp. 178–185.

[7] J. Cong, C. Koh, and K. Leung, "Simultaneous buffer and wire sizing for performance and power optimization," in *Proc. Int. Symp. Low Power Electron. Design*, 1996, pp. 271–276.

[8] C.-P Chen, C. C. N. Chu, and D. F. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1998, pp. 617–624.

[9] T. Okamoto and J. Cong, "Buffered Steiner tree construction with wire sizing for interconnect layout optimization," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1996, pp. 44–49.

[10] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion with accurate gate and interconnect delay computation," in *Proc. 36th ACM/IEEE Design Automation Conf.*, 1999, pp. 479–484.

[11] L. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp. 865–868.

[12] I.-M Liu, T.-L Chou, A. Aziz, and D. F. Wong, "Zero-skew clock tree construction by simultaneous routing, wire sizing, and buffer insertion," in *Proc. Int. Symp. Physical Design*, 2000, pp. 33–38.

[13] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, and A. Kahng, "Zero skew clock routing with minimum wirelength," *IEEE Trans. Circuits Syst. II*, vol. 39, pp. 799–814, Nov. 1992.

[14] J.-L. Tsai, T.-H. Chen, and C. C.-P. Chen, "Epsilon-optimal minimum-delay/area zero-skew clock-tree wire-sizing in pseudo-polynomial time," in *Proc. Int. Symp. Physical Design*, 2003, pp. 166–173.

[15] S. Pullela, N. Menezes, J. Omar, and L. T. Pillage, "Skew and delay optimization for reliable buffered clock trees," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1993, pp. 556–562.

[16] P. R. O'Brien and T. L. Savarino, "Modeling the driving-point characteristic of resistive interconnect for accurate delay estimation," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1989, pp. 512–515.

[17] X. Tang, R. Tian, H. Xiang, and D. F. Wong, "A new algorithm for routing tree construction with buffer insertion and wire sizing under obstacle constraints," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 2001, pp. 49–56.

[18] R.-S. Tsay, "Exact zero skew," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1991, pp. 336–339.

Constrained Floorplanning Using Network Flows

Yan Feng, Dinesh P. Mehta, and Hannah Yang

***Abstract*—This paper presents algorithms for a constrained version of the "modern" floorplanning problem proposed by Kahng in "Classical Floorplanning Harmful?" (Kahng, 2000). Specifically, the constrained modern floorplanning problem (CMFP) is suitable when die-size is fixed, modules are permitted to have rectilinear shapes and, in addition, the approximate relative positions of the modules are known. This formulation is particularly useful in two scenarios: 1) assisting an expert floorplan architect in a semiautomated floorplan methodology and 2) in incremental floorplanning. CMFP is shown to be negative–positive hard. An algorithm based on a max-flow network formulation quickly identifies input constraints that are impossible to meet, thus permitting the floorplan architect to modify these constraints. Three algorithms [Breadth First Search (BFS), Improved BFS (IBFS), Compromise BFS (CBFS)] based on using BFS numbers to assign costs in a min-cost max-flow network formulation are presented. Experiments on standard benchmarks demonstrate that IBFS is fast and effective in practice.**

***Index Terms*—Algorithms, design automation, flow graphs.**

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. The objective is to minimize some combination of the area, estimated wire length, and other criteria that have emerged recently such as critical-path wire length, length of parallel-running wires, clock skew, etc. Much research in floorplanning is concerned with finding a good representation that can be used efficiently within the context of simulated annealing [2]–[9].

Kahng [1] critiques the classical floorplanning problem and proposes a modern formulation that is more consistent with the needs of current design methodologies. Some of the attributes of the modern 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 methodologies; 2) the modules' shapes should not be restricted to rectangles, L-shapes, and T-shapes; and 3) "round" blocks with an aspect ratio near 1 are desirable.

Several aspects of this problem had been previously addressed by Mehta and Sherwani [10]. Their algorithm assumes a fixed outline and obtains a provable zero whitespace solution by relaxing the requirement on module shapes. Further, it also tries to make blocks as "round" as possible and to minimize the number of sides. Their methodology differs from that proposed by Kahng in that they assume that an approximate location for each module was included in the input. This is a realistic formulation in several design scenarios where the designer already has a fairly good idea as to the approximate locations of the modules. This claim is supported by an excerpt reproduced from a discussion among designers in an electrical design automation (EDA) newsgroup:

Manuscript received May 29, 2003; revised September 19, 2003. This work was supported by the National Science Foundation under Grant CCR-9988338. This paper was recommended by Guest Editor C. J. Alpert.

Y. Feng and D. P. Mehta are with the Department of Mathematical and Computer Science, Colorado School of Mines, Golden, CO 80401 USA (email: yfeng@mines.edu; e-mail:dmehta@mines.edu).

H. Yang is with the Strategic Computer-Aided Design Labs, Intel, Hillsboro, OR 97124 USA (e-mail:hyang@ichips.intel.com).

Digital Object Identifier 10.1109/TCAD.2004.825877

As the designer, you already know (or should have a good idea) of how you want data to physically flow in the device. As in, take the inputs from one corner of the device, do computation in the middle, and then output on one of the other corners.

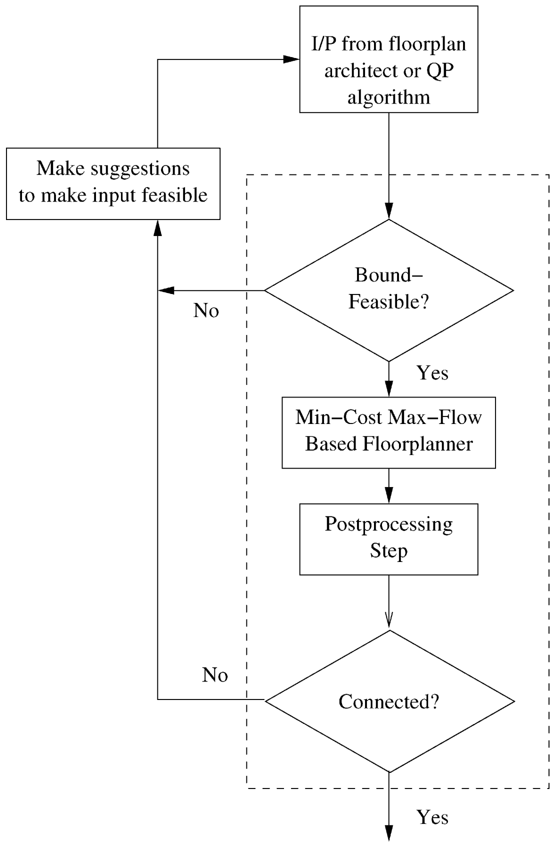
The designer may have designed a very similar chip in the past and may wish to reuse the approximate relative locations from the previous design. Yet another scenario where this formulation will find application is in *incremental floorplanning*. In this scenario, a floorplan has already been obtained by traditional means. However, the sizes of some of the modules have been changed and we need to compute a new floorplan that respects the relative locations of the initial floorplan. Incremental floorplanning was studied in [11], but this was limited to growing modules into the adjoining whitespace, if such whitespace was available.

One key benefit of this formulation of the problem is that the relative locations of modules have been computed and *criteria such as wire length, congestion, etc., have already been considered and optimized*. This makes it possible to develop algorithms that do not have to take these factors into account. Another benefit is that deterministic algorithms (i.e., as opposed to simulated annealing) can be employed. In this paper, we make significant improvements to the solution in [10]. First, the modules obtained by [10] can have "snake-like" stringy shapes and the distance between the farthest pair of points in the module can be large. This is because modules are placed sequentially with the result that the last few blocks are severely constrained by the available area. Second, the algorithm dissects the floorplan area into a grid, resulting in a complexity of $O(A^2)$, where A is the area of the floorplan.

In this paper, we formulate the problem differently *so that the distance between the farthest pair of points in each module is bounded*. We show that the problem is negative–positive (NP)-hard by exploiting its relationship to a scheduling problem. A max-flow algorithm determines whether the input satisfies feasibility criteria. A min-cost max-flow heuristic is used to compute the floorplan. Our network flow-based solutions take a global and integrated view that simultaneously includes all modules and is an improvement over the [10], which floorplanned modules sequentially. Unlike [10], connectivity cannot be guaranteed. However, our experiments show that module connectivity was achieved on all benchmarks. The modules computed by our algorithm are compact (i.e., they do not have stringy shapes) and have fewer sides than those in [10], even though we are using larger and more complex benchmarks than those used in [10]. Although the worst case complexity of our algorithm is high ($O(n^6 \log n)$), where n is the number of modules, we believe that this only applies to pathological situations that our algorithm is not likely to encounter in a real environment. A more realistic worst-case complexity is shown to be $O(n^2 \log^2 n)$. This is supported by our experimental results and is an improvement over [10] which has a pseudo-polynomial time complexity.

The fixed-die aspects of Kahng's formulation have also been addressed recently by Adya and Markov in [12] and [13], and by Tang and Wong [14]. However, they restrict module shapes to be rectangles, and information about the relative locations of modules is not required. Simulated annealing is used to optimize the floorplan.

Fig. 1 describes our design flow. In a highly constrained problem such as this, it is possible that a given input is infeasible. Thus, we carry out two feasibility checks. The first check (*bound-feasibility*) ensures that it is possible for all modules to be assigned area within the given bounds. An input passes the bound-feasibility check if and only if it is bound-feasible. If this is the case, we proceed with our floorplanning algorithm that assigns area to each module. This is followed by a postprocessing step that makes minor changes to the area assignment so that the area assigned to each module is connected. This is followed by a *connectivity* check. If either of the checks fails, the appropriate feedback will be provided to the input-generation step (the user) along with recommendations on how to minimally change the input to make it feasible. This paper is concerned with the steps enclosed in the dashed box in the flowchart. The user-feedback step will be the subject of a future paper. There is a possibility that our algorithm will reject a feasible input if the postprocessing step is unable to make all the modules connected. However, since the problem is NP-hard, it is unlikely that a fast algorithm exists that guarantees a connected solution for every feasible input. All of our experiments with bound-feasible inputs showed that the postprocessing step was able to compute valid connected floorplans. (In other words, if our input passed the bound-feasibility check, it also passed the subsequent connectivity check.)


I/P from floorplan architect or QP algorithm
Make suggestions to make input feasible
Bound–Feasible?
No
Yes
Min–Cost Max–Flow Based Floorplanner
Postprocessing Step
No
Connected?
Yes
Fig. 1. Design flow context for this paper. The steps contained in the large dashed rectangle are addressed in this paper.

Section II defines constrained modern floorplanning problem (CMFP) and proves that it is NP-hard. Section III describes an algorithm that determines bound-feasibility. Section IV describes algorithms that assign regions to modules. Section V discusses the postprocessing step and provides experimental results.

II. CMFP

INPUT: 1) An $H \times W$ bounding box that denotes the fixed die and 2) aset of modules N such that each module $m_i \in N, 1 \leq i \leq n = |N|$, is denoted by a quintuple $(x_i, y_i, r_i^x, r_i^y, A_i)$. 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 to be assigned to module m_i. r_i^x and r_i^y 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 $[x_i - r_i^x, x_i + r_i^x]$ in

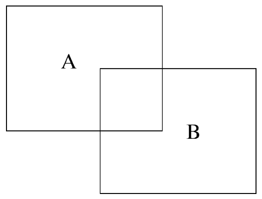


Fig. 2. Two constraining rectangles A and B each have an area of 20 units. The area of $A \cap B$ is four units. However, the modules that must be contained in A and B both require 19 units. Thus, 38 units of area must be accommodated in a constrained area of 36 units, which is not possible.

the x-direction and $[y_i - r_i^y, y + r_i^y]$ in the y-direction, and 3) m_i must be connected.

Observe that this formulation will handle hard, preplaced modules (i.e., modules with fixed dimensions and locations) by letting (x_i, y_i) denote the coordinates of the desired center of module m_i, choosing r_i^x and r_i^y to be half the width and height, respectively, of the module, and setting $A_i = 4r_i^x r_i^y$.

Theorem 1: CMFP is NP-hard

Proof: Consider the one-dimensional special case of CMFP (CMFP_1D). The input to CMFP_1D consists of a bounding interval [0, W] on the x-axis. Each module has an associated length l_i (replacing A_i from CMFP) and a bounding interval $[x_i - r_i^x, x_i + r_i^x]$. The question is whether it is possible to allocate exactly one interval (from the connectivity requirement) to each module of length l_i within its bounding interval (and also within [0, W]). Observe that this is a restatement of Problem [SS1] **SEQUENCING WITH RELEASE TIMES AND DEADLINES** in [15, p. 236], which is known to be NP-complete. Thus, CMFP_1D and CMFP are also NP-hard. ■

III. Flow-Based Bound-Feasibility Computation

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

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

$$\sum_{m_i \in T} A_i \le \text{Area}(\cup_{m_i \in T} Q_i). \tag{1}$$

Fig. 2 shows an example of an infeasible input. (See Fig. 11 for another example.)

Next, we develop a max-flow-based algorithm for checking bound-feasibility. The constraining rectangles dissect 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.

Define the flow network $G = (V, E)$ as follows. Let $V = \{s, t\} \cup L \cup R$, where s is the source and t is the sink. Each vertex in L corresponds to a constraining rectangle. Each vertex in R represents a region. Let $E = E_1 \cup E_2 \cup E_3$, where

1) $E_1 = \{(s, u) : u \in L\}$. Recall that u corresponds to a constraining rectangle. The capacity of edge (s, u) is A_u.
2) $E_2 = \{(u, v) : u \in L$ and $v \in R$, such that constraining rectangle u contains region $v\}$. The capacity of edge (u, v) is the area of region v.
3) $E_3 = \{(v, t) : v \in R\}$. The capacity of edge (v, t) is the area of region v.

Fig. 4 shows the flow graph corresponding to Fig. 3.

Theorem 2: If the maximum network flow of G is equal to the total required area of the modules ($\sum_{i=1}^n A_i$), then the input is bound-feasible.

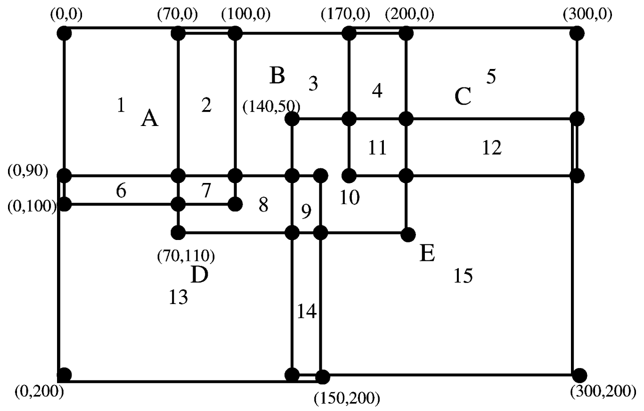


Fig. 3. Five rectangles A–E dissect the bounding rectangle into 15 faces or regions. Different colors/shades have been used for different rectangles. Rectangle labels have been located approximately at their centers. The shaded circles correspond to vertices of the dissection graph. In this example, A is a red rectangle consisting of Regions 1, 2, 6, and 7; B is the yellow rectangle consisting of Regions 2–4 and 7–11; C is a red rectangle consisting of 4, 5, 11, and 12; D is the blue rectangle and consists of Regions 6–9, 13, and 14; and E is the green rectangle consisting of Regions 9–12, 14, and 15. Collinear edges have been staggered for clarity.

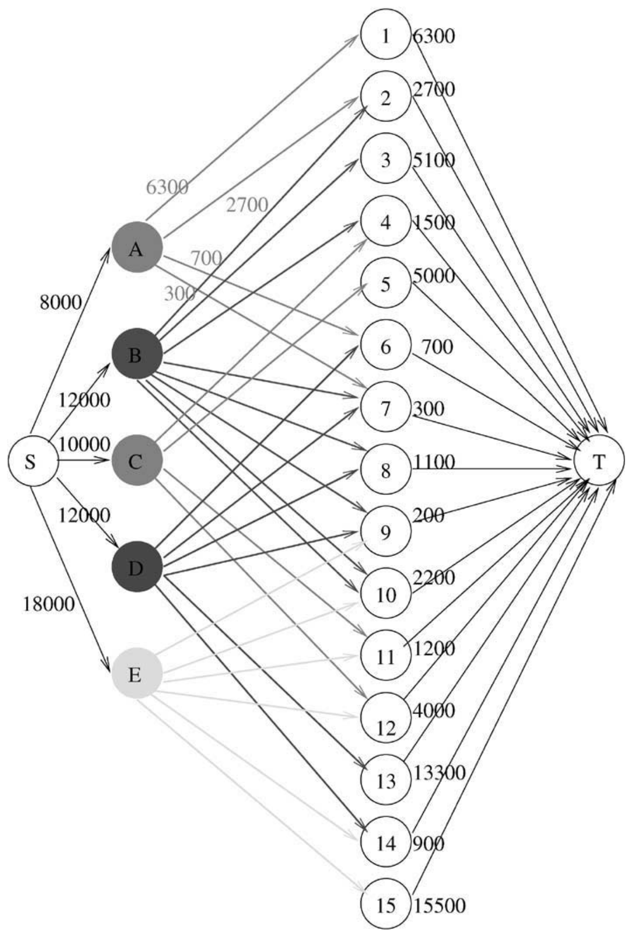


Fig. 4. Flow graph corresponding to Fig. 3 (**best seen in color**).

Proof: Let f be the maximum flow through G. The cut $C_1 = (\{s\}, L \cup R \cup \{t\})$ has capacity $\sum_i A_i$. So, $f \le \sum_i A_i$. Consider an edge (u, v) in E_2. We interpret $f(u, v)$ to mean the assignment of $f(u, v)$ units of area in region v to rectangle u. Clearly, rectangle

u cannot be assigned area greater than A_u because of flow conservation at vertex u. Similarly, no region v can contribute an area greater than Area(v) because of flow conservation at v. So, $\sum_{v \in R} f(u,v) \leq A_u$ and $\sum_{u \in L} f(u,v) \leq$ Area(v). All modules have been legally assigned their required areas and the input is feasible if and only if $f = \sum_i A_i$. ■

Lemma 1: $|R|$ is $O(n^2)$.

Proof: The input rectangles dissect the bounding rectangle into faces or regions. We can view the dissection as a planar graph: the vertices of the graph consist of the four vertices of each input rectangle and the points of intersection between a pair of orthogonal sides (See Fig. 3). Note that the number of intersections is bounded by $4n^2$ as there are $2n$ vertical sides and $2n$ horizontal sides. So v, the number of vertices, is $4n + 4n^2$. The edges of the graph correspond to vertical or horizontal line segments that join a pair of vertices. If no two rectangles intersect, the number of edges would be $4n$. However, each point of intersection results in at most two additional edges. Thus, the maximum possible number of edges e is $4n + 8n^2$. From Euler's formula for planar graphs, we have $v - e + f = 2$. Thus, $|R| = f = O(n^2)$. ■

We next define the intersection graph $IG(V, E)$ of a set of rectangles A_i. Each vertex of IG corresponds to a rectangle. There is an edge between a pair of vertices if and only if the corresponding rectangles intersect.

Lemma 2: If the rectangle intersection graph on n rectangles has $O(n)$ edges, then $|R|$ for the dissection graph induced by the rectangles is also $O(n)$.

Proof: When two rectangles intersect, the number of additional vertices created in the dissection graph is 4. So, if there are $O(n)$ intersections, then the total number of vertices in the dissection graph is also $O(n)$. An argument similar to that in the proof for Lemma 1 shows that the number of edges in the dissection graph is also $O(n)$. From Euler's formula, we can then conclude that $|R| = O(n)$. ■

Lemma 3: If each region is contained in a constant number of modules, then the number of edges in the flow graph is $O(|R|)$.

Proof: $|E_1| = n \leq |R|$ and $|E_3| = |R|$. Since each region in R is contained in a constant number of modules, the number of incoming edges from vertices in L to a vertex in R is bounded by a constant and $|E_2| = O(|R|)$. ■

Our experimental observations confirm that in most examples, the number of edges in the intersection graph is $O(n)$. Combining this with Lemmas 2 and 3, we conclude that the number of vertices and edges in the flow graph are $O(n)$ in practice.

Theorem 3: The feasibility algorithm requires $O(n^5 \log n)$ time in the worst case and $O(n^2 \log n)$ in practice.

Proof: An $O(n^2)$ planesweep algorithm is used to identify the regions of the dissection. The maximum flow algorithm runs in $O(|V||E| \log |V|^2/|E|)$ time [16]. In the worst case, $|V| = O(n^2)$ and $|E| = O(n^3)$, resulting in a complexity of $O(n^5 \log n)$. In practice, we expect the number of intersections to be $\Theta(n)$, giving $|V| = \Theta(n)$ (Lemma 2). From Lemma 3, we have $|E| = \Theta(n)$. Under these assumptions, the runtime is $O(n^2 \log n)$. ■

IV. Constraint-Based Floorplanning Algorithm

The flow-based feasibility algorithm of the previous section can be used to assign regions to modules so as to assure a zero whitespace floorplan. However, a module could be assigned area in regions that are not adjacent to each other. For example, in Fig. 5, which shows a constraining rectangle and its six regions, the module could be assigned areas in Regions 2, 4, and 6 which disconnects the module. This violates one of the constraints of CMFP. In this section, we rectify this by employing a min-cost max-flow algorithm on the flow graph of the previous section. This seeks to minimize the sum of the product of edge

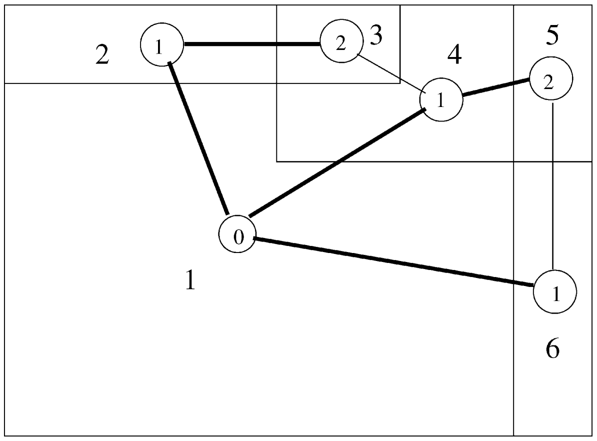


Fig. 5. Constraining rectangle with six regions. A region graph with six vertices and edges between adjacent regions is also shown. BFS originates in Region 1 which contains the center of the constraining rectangle. BFS numbers are shown inside the vertices. Thicker edges are edges in the breadth first tree that were used to assign labels.

costs and edge flows while maintaining maximum flow through the network. The key idea is that we have to assign costs to edges in a way that encourages the assignment of *connected* regions to modules.

A. BFS Algorithm

BFS numbering is a method for assigning costs to the regions in a module: the region containing the center of the constraining rectangle is labeled 0. Suppose labels 0 through i have been assigned to some regions for $i \geq 0$. Then, all unlabeled regions that share a portion of a side with regions labeled i are labeled $i + 1$. The process is repeated until all regions in the module are labeled. An alternative description uses the notion of a region graph. Each region of a constraining rectangle m is represented by a vertex. Two vertices are joined by an edge if and only if the corresponding regions share a common segment on their boundaries. Let s be the vertex corresponding to the region that contains the center of the constraining rectangle. The BFS label of a region r described above is equivalent to the number of edges in the shortest path from s to r in the region graph and is obtained by performing a BFS traversal [16] initiated at s. The breadth first number computed for each region r is used to assign a cost to the edge in the flow graph from module m to region r. Fig. 5 illustrates these ideas.

The philosophy of this approach is that regions will be "filled" in increasing order of cost by the min-cost algorithm, i.e., from the center toward the periphery, resulting in connected blocks. However, there are some scenarios such as in Fig. 6 where the algorithm cannot find connected solutions, even though they may exist.

B. Improved BFS (IBFS) Algorithm

BFS numbering results in edge costs being integers. This creates several scenarios where two or modules competing for the same region have identical priorities (costs). The min-cost max-flow algorithm then assigns area to the modules in an arbitrary fashion. In the improved-BFS (IBFS) algorithm, we assign fractional costs to edges making IBFS more discriminating than BFS. In IBFS, we address the situation when edges (l_i, r) and (l_j, r) have the same BFS numbers in the flow graph, where l_i, $l_j \in L$ and $r \in R$. In BFS, this permitted the min-cost max-flow algorithm to (somewhat) arbitrarily assign the area of region r to either l_i or l_j. In IBFS, we address this scenario as follows: consider modules A and B in Fig. 3. Their constraining rectangles have areas 10 000 and 14 300, respectively. A must be assigned 8000 units and B must be assigned

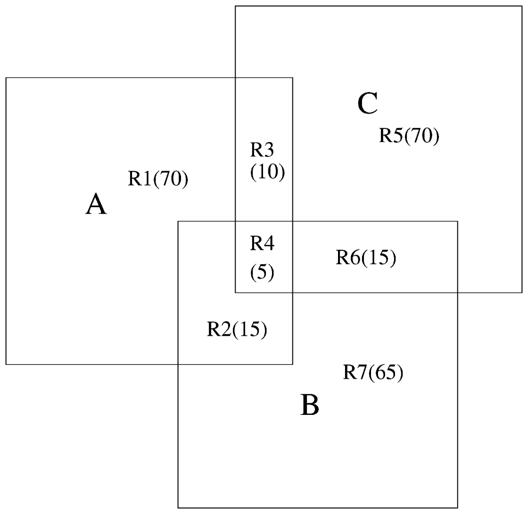


Fig. 6. Example showing that BFS may not yield a connected solution even when a connected solution exists: modules A, B, and C require 75, 80, and 95 area units, respectively. Assume that module C gets its required area from $R3(10)$, $R5(70)$, and $R6(15)$. A has exclusive access to 70 units of area in $R1$ and B has exclusive access to 65 So A needs five additional units and B needs 15 additional units from $R2$ and $R4$. Edges $(A, R2)$ and $(B, R2)$ both have cost 1. Edges $(A, R4)$ and $(B, R4)$ both have cost 2. The min-cost max-flow algorithm might then assign R2 to B and R4 to A, making module A disconnected.

12 000 units. Consider Region 2 which lies in $A \cap B$ with area 2700. In BFS, edges $(A, 2)$ and $(B, 2)$ would be assigned a cost of one unit. In IBFS, we take into account that A has exclusive access to the 6300 units of Region 1 and B has exclusive access to the 5100 units of Region 3. Thus, A needs an additional 3700 units from shared regions and B needs an additional 7200 units. These quantities are used to assign a cost of $1 + (7200/3700 + 7200)$ to edge $(A, 2)$ and a cost of $1 + (3700/3700 + 7200)$ to edge $(B, 2)$. In other words, we add a fractional cost inversely proportional to the relative additional area required by each module to the cost generated in BFS. This cost-modification procedure is applied to every group of modules that share a region *and* were assigned the same costs by BFS. To compute the additional area required for regions for which modules have a BFS cost greater than 1, we assume that all shared regions with lower costs were distributed equally among the contending regions.

C. Compromise-BFS (CBFS) Algorithm

In CBFS, a region is conceptually split into a number of subregions equal to the number of modules competing for the region. Each subregion is associated with a module, which is given highest priority (i.e., the least cost edge) among the competing modules. The remaining modules continue to have access to the subregion, but with lower priority. This ensures that all modules are assigned some area in the region, provided that a feasible max-flow solution is obtained. More formally, if a region r is shared by modules $m_1, \ldots, m_k$, we replace the vertex corresponding to region r in the flow graph with vertices $r_1, \ldots, r_k$. An edge is created between each module in m_i, $1 \leq i \leq k$ and every region r_i, $1 \leq i \leq k$. The edge between m_i and r_i is assigned its original cost from BFS. All the other edges have cost c, which is a large constant. Fig. 7 shows an example involving modules B, C, and E and Regions 4, 5, 11, and 12 from Fig. 3.

Theorem 4: The computational complexity for BFS and IBFS is $O(n^6 \log n)$ in the worst case and is $O(n^2 \log^2 n)$ in practice.

Proof: An $O(n^2)$ planesweep algorithm is used to identify regions and their adjacencies. This provides the necessary information to construct the flow graph. A minimum-cost maximum-flow algorithm using enhanced capacity scaling [17] requires $O(|E| \log |V|(|E| + |V| \log |V|))$ time, where $|V|$ is the number of vertices and $|E|$ is the number of edges. In the worst case, $|V| = O(n^2)$ and $|E| = O(n^3)$, giving $O(n^6 \log n)$. As discussed in Section III, we expect $|E| = |V| = \Theta(n)$ in practice, giving $O(n^2 \log^2 n)$. ■

Theorem 5: The computational complexity for CBFS is $O(n^8 \log n)$ in the worst case and is expected to be $O(n^2 \log^2 n)$ in practice.

Proof: In the worst case, $|V| = O(n^3)$ and $|E| = O(n^4)$, giving $O(n^8 \log n)$. We expect $|E| = |V| = \Theta(n)$ in spite of duplicating regions because regions are typically contained in a constant number of modules. Thus, $|V|$ and $|E|$ will only increase by a constant factor, giving the same results as before. ■

V. Postprocessing Step and Experimental Results

A. Postprocessing Step

Although the output of the three BFS-based algorithms specify how much area of a region should be assigned to a module, they do not specify exactly how to assign a geometric area to each module. Furthermore, since CMFP is NP-hard, it is unlikely that a fast algorithm can find a connected solution even if such a solution exists. The postprocessing steps outlined in this section are concerned with making modules connected and assigning an exact geometric area to each module.

The output of any of the BFS algorithms can be represented in the region graph (described in the previous section) RG as follows: each region vertex contains a list of the modules that were *allocated* area in that region by the BFS algorithm along with the area that was allocated to each. A module m is typically allocated area from several regions. Let this subset of regions be called R_m. Module m is *graph-connected* if and only if the subgraph of RG, induced by vertices corresponding to R_m, is connected. We will say that a module-region assignment (the output of a BFS algorithm) is graph-connected if and only if each of the modules is graph-connected. Clearly, the following apply.

Lemma 4: A floorplan can be connected only if the corresponding module-region assignment was graph-connected.

Our experiments show that after running the BFS-based algorithms: 1) few modules are disconnected; 2) if a module is disconnected, it typically consists of one large connected component and few very small connected components; and 3) the connected areas are near each other and are separated by at most one intervening region. The only scenario with disconnected modules that we encountered is similar to that illustrated in Fig. 8. Thus, our postprocessing step simply exchanges module-assignments among neighboring regions to make the output graph-connected.

Lemma 5: A graph-connected module-region assignment does not imply the existence of a connected floorplan (See Fig. 9).

Our geometric-connectivity postprocessing step only considers regions that will be shared by several modules and floorplans such regions by considering their neighborhood in RG. This can be simplified considerably if the number of modules that are to be assigned to the same region is small. Thus, CBFS, which generates outputs where the number of modules sharing a region can be quite high (as the experiments will show) is not a viable algorithm. Although there is still a theoretical possibility that geometric connectivity cannot be achieved, we did not encounter such a case in BFS and IBFS, presumably because of the clustered nature of the region-module assignments resulting from the BFS-based algorithms. Fig. 10 shows a typical scenario addressed by our geometric postprocessing step.

B. Experiments

The planesweep algorithm that identifies regions, their areas, and adjacencies between regions was implemented in Java. The

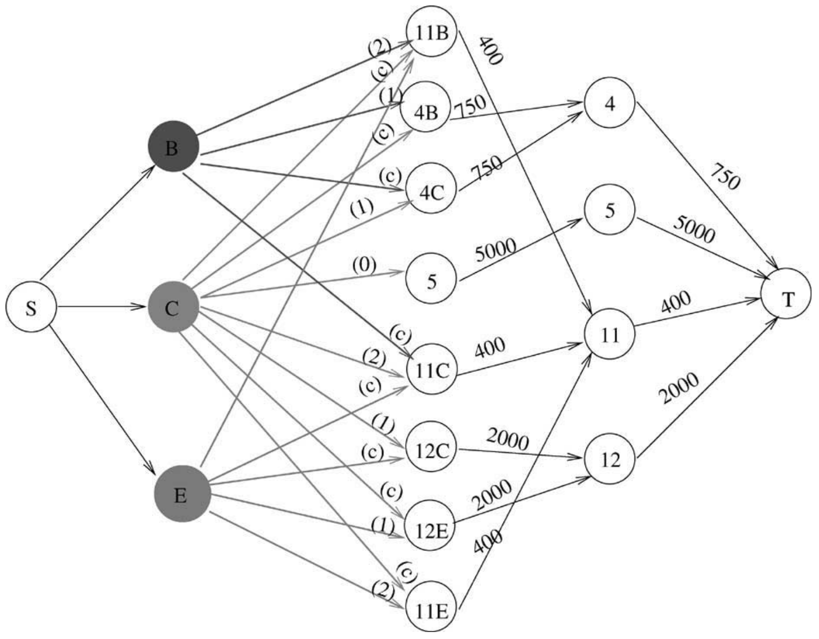


Fig. 7. Partial flow graph for CBFS. Observe that Region 4 (1500 units) is shared by modules B and C in Fig. 3. We split vertex 4 into 4B and 4C (750 units each). The costs of edges $(B, 4B)$ and $(C, 4C)$ are the same as in BFS. Edges $(B, 4C)$ and $(C, 4B)$ are now assigned a large constant c.

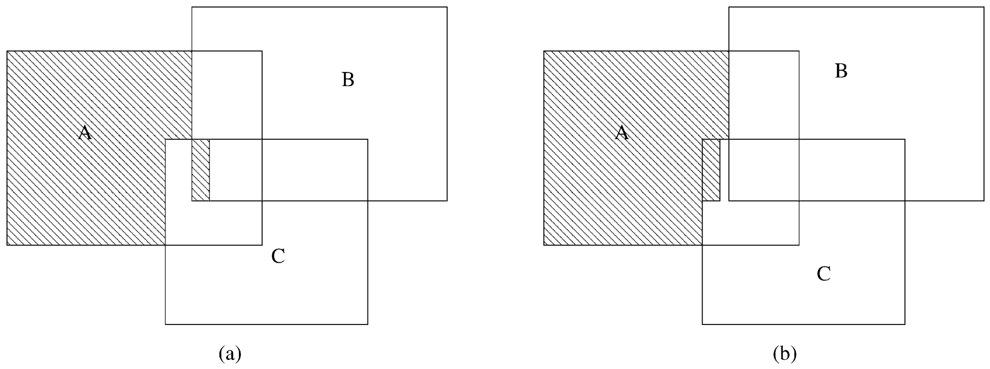


Fig. 8. Disconnected module. (a) Module A is disconnected. However, this can be rectified as shown in (b) by swapping area with module C.

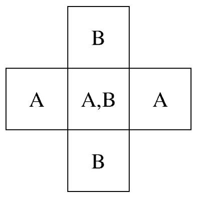


Fig. 9. Five regions and the modules that will be assigned to them. Both modules A and B are graph-connected. However, if A is connected in the floorplan, then B is not (and vice versa). This illustrates that graph connectivity does not imply geometric connectivity.

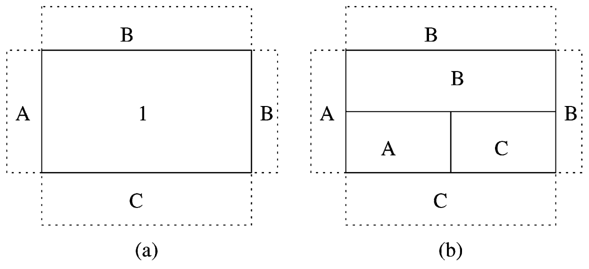


Fig. 10. (a) Region 1 is surrounded by regions that have been assigned areas to modules A, B, and C. Region 1 has itself been assigned to modules A, B, and C. The geometric postprocessing step determines how the real estate in Region 1 will be allocated to A–C. (b) shows that A–C are allocated real estate so that each module is assigned contiguous areas next to the appropriate adjoining region.

infeasibility-detection algorithm of Section III and the min-cost max-flow algorithm of Section IV were implemented on a Linux workstation running on a Pentium processor in C++ using LEDA [18]. The results presented here are based on Microelectronics Center of North Carolina benchmarks ami33 and ami49, which were adapted for use with our algorithms. We also used Gigascale Systems Research Center (GSRC) benchmark n300a [19] to verify the applicability of our algorithms to larger designs. A simulated annealing step with a cost function that emphasized wirelength minimization (over area minimization) was used to obtain initial positions for our modules.

TABLE I
INPUT CHARACTERISTICS. BENCHMARKS AMI33 (33 MODULES) AND AMI49 (49 MODULES) WERE USED. CR RATIO IS USED TO DETERMINE THE DIMENSIONS OF THE CONSTRAINING RECTANGLE FOR EACH MODULE IN THE PARTICULAR INPUT. SO, A $w \times h$ MODULE IN I_1 HAS A CONSTRAINING RECTANGLE OF DIMENSION $1.4w \times 1.4h$. OBSERVE THAT THE NUMBER OF REGIONS IN THE DISSECTION INCREASES AS THE CR RATIO INCREASES

Input ID	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
Benchmark	ami33	ami33	ami33	ami33	ami33	ami49	ami49	ami49	ami49	ami49
CR ratio	1.4	1.5	1.6	1.7	1.8	1.4	1.5	1.6	1.7	1.8
No of regions	298	338	393	414	426	448	469	508	542	564

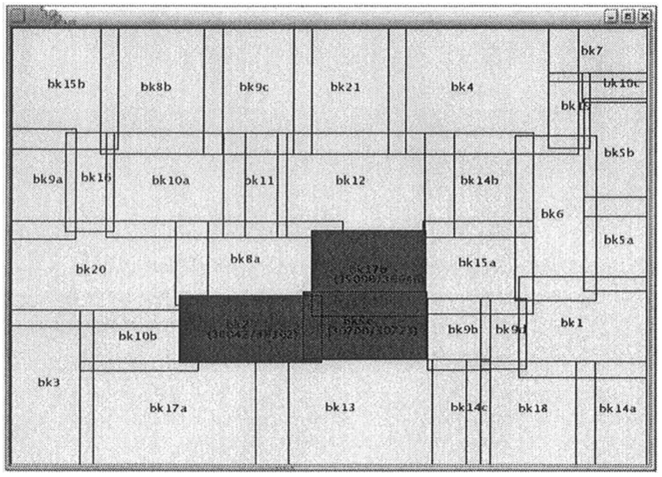


Fig. 11. Bound-infeasible floorplan. Dark (red, if viewed in color) constraining rectangles correspond to blocks that were unable to get the required area. The output displays the allocated area divided by the required area for these rectangles.

TABLE II
NUMBER OF DISCONNECTED REGIONS. EACH MODULE IS ASSIGNED SOME REGIONS. TYPICALLY, EACH MODULE HAS ONE LARGE COMPONENT CONSISTING OF SEVERAL REGIONS. IF A MODULE IS DISCONNECTED, WE COUNT HOW MANY REGIONS OF THE MODULE ARE NOT PART OF THIS LARGE COMPONENT. THE SUM OF THIS QUANTITY OVER ALL MODULES IN THE FLOORPLAN GIVES THE NUMBER OF DISCONNECTED REGIONS. OBSERVE THE IMPROVEMENTS OBTAINED BY OUR MIN-COST BASED ALGORITHMS RELATIVE TO MAX-FLOW

Input ID	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
MaxFlow	32	50	58	70	67	20	34	38	48	46
BFS	9	2	13	5	5	6	6	13	5	11
IBFS	7	9	3	11	2	2	3	5	4	5
CBFS	0	0	0	0	0	0	0	0	0	0

They were perturbed to accommodate constraining rectangles within the specified floorplan bounding box.[1] Constraining rectangles were chosen as described in Table I.

Observe that CR ratios between 1.4 and 1.8 were used. Our experiments with smaller CR Ratios such as 1.1 and 1.2 overconstrained the problem and resulted in infeasible inputs. Fig. 11 shows the output of our infeasibility-detection algorithm, when it was run on an overconstrained adaptation of ami33.

Tables II and III present metrics comparing our algorithms for assigning regions to modules *before postprocessing*. These metrics determine whether the algorithms produce an output that is amenable to postprocessing. Algorithms that do not perform well, with respect to these metrics, were discarded. Thus, MaxFlow was discarded after Table II because of the large number of disconnects. CBFS was discarded after Table III because of the large number of modules sharing a region.

[1]Although our algorithms are more appropriately preceded by quadratic or force-directed placement because they result in overlapping modules, we did not use these methods as these placement tools were not available when we began this research. We recently noticed that a force-directed placement tool is available at GSRC and plan to integrate its results into our flow.

TABLE III
EACH ENTRY SHOWS THE RESULT WHEN A PARTICULAR ALGORITHM IS RUN ON A PARTICULAR INPUT. THE VALUE OUTSIDE THE PARENTHESIS IS THE MAXIMUM NUMBER OF MODULES THAT ARE AASSIGNED AREA IN ANY REGION OF THE FLOORPLAN. THE NUMBER INSIDE THE PARENTHESIS IS THE NUMBER OF REGIONS THAT SHARE THIS MAXIMUM NUMBER OF MODULES. THUS, THE ENTRY CORRESPONDING TO THE EXECUTION OF BFS ON I_1 INDICATES THAT THERE WERE EIGHT REGIONS THAT WERE ASSIGNED THREE MODULES AND THAT NO REGIONS WERE ASSIGNED FOUR OR MORE MODULES. OBSERVE THAT THE NUMBERS FOR BFS AND IBFS ARE SUPERIOR TO THOSE FOR CBFS

Input ID	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
BFS	3(8)	3(5)	3(2)	3(4)	3(9)	3(7)	4(1)	3(7)	4(1)	3(7)
IBFS	3(3)	3(1)	3(2)	3(2)	3(1)	3(4)	3(3)	3(2)	3(2)	3(4)
CBFS	7(4)	8(1)	8(3)	8(3)	8(4)	8(5)	8(2)	8(1)	8(3)	8(3)

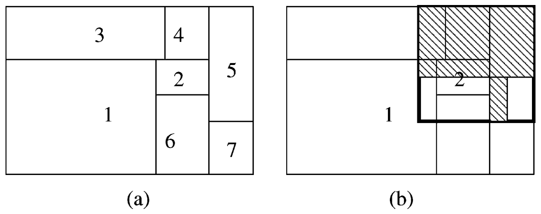


Fig. 12. Center adjustment step. (a) Single constraining rectangle and its regions. The center is located in Region 1 and is the starting region for the initial breadth first search algorithm. The shaded area in (b) is the area that is actually assigned to the module following the postprocessing step. The center of the bounding box of this area falls inside Region 2. The center adjustment step consists of now rerunning the BFS numbering algorithm starting at Region 2.

TABLE IV
NUMBER OF DISCONNECTED REGIONS BEFORE AND AFTER CENTER ADJUSTMENT FOR IBFS

Input ID	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
Before Center Adjustment	7	9	3	11	2	2	3	5	4	5
After Center Adjustment	3	5	3	1	4	2	1	4	1	5

Tables IV and V display the same metrics as Tables II and III, respectively, for the IBFS algorithm. They show that additional improvements in the metrics may be obtained by carrying out a *center adjustment* step. Center adjustment is concerned with the region from which the BFS numbering is initiated (i.e., the region that gets assigned a cost of 0). This is initially chosen to be a region toward the center of the constraining rectangle. However, after running IBFS, it is often the case that modules have been assigned the bulk of their area in a corner of the constraining rectangle. We then determine approximately the center of the floorplanned module and rerun IBFS using the new (improved) center for the BFS numbering. Fig. 12 illustrates this step. We found that center adjustment often improved the solution quality as shown in Tables IV and V.

TABLE V
MAXIMUM NUMBER OF MODULES THAT ARE ASSIGNED TO A REGION AND, IN PARENTHESES, THE NUMBER OF REGIONS WITH A MAXIMUM-MODULE ASSIGNMENT. THESE ARE SHOWN BEFORE AND AFTER CENTER ADJUSTMENT FOR IBFS. OBSERVE THAT THE ENTRY FOR I_7 AFTER ADJUSTMENT IS $2(*)$. THIS INDICATES THAT THERE WERE NO REGIONS SHARED BY THREE OR MORE MODULES. THE * INDICATES THAT THE NUMBER OF REGIONS WITH TWO MODULES IS NOT OF INTEREST

Input ID	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
Before Center Adjustment	3(3)	3(1)	3(2)	3(2)	3(1)	3(4)	3(3)	3(2)	3(2)	3(4)
After Center Adjustment	3(1)	3(2)	3(3)	3(1)	3(4)	3(1)	2(*)	3(2)	3(2)	3(5)

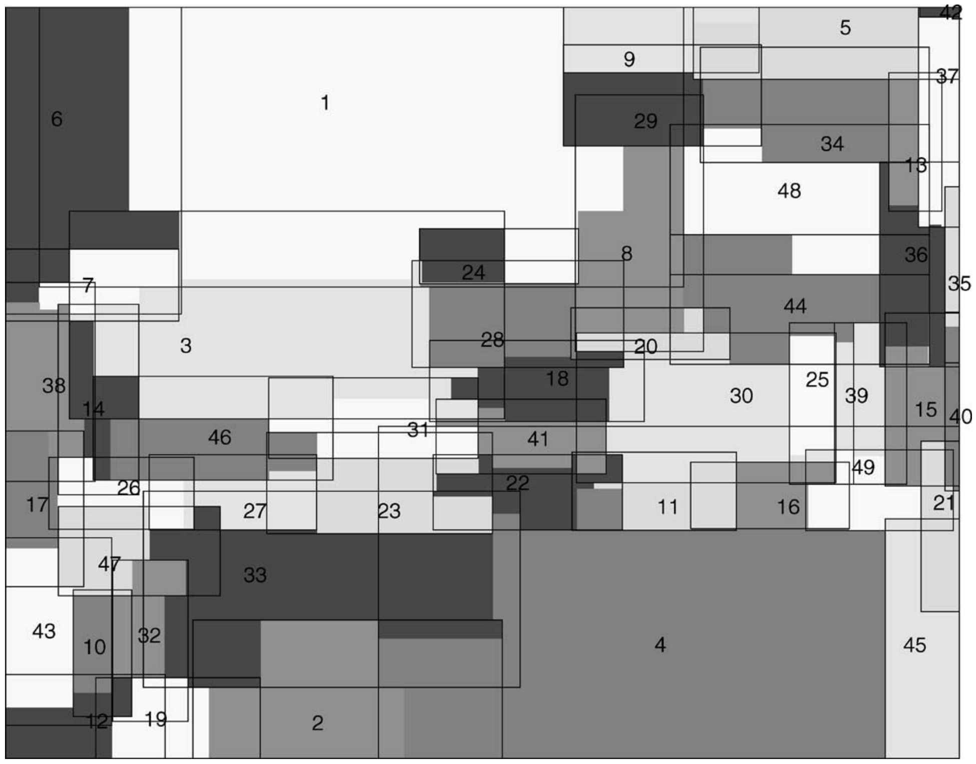


Fig. 13. Results of IBFS, followed by postprocessing on input I_6. Constraining rectangles and actual assignment of areas to modules are shown. The floorplan has zero whitespace and all modules are connected and are within their bounds. (Best viewed in color.)

TABLE VI
NUMBER OF SIDES FOR IBFS. NOTICE THAT THIS QUANTITY INCREASES AS THE CR RATIO INCREASES

Input ID	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
Avg No of Sides	12.5	13.5	14.6	15.3	16.1	14	14.5	14.9	15.3	15.8

Next, the postprocessing step was run on outputs of BFS and IBFS. Modules were restricted to their constraining rectangles, were connected, and zero-whitespace floorplans were obtained in every case. Fig. 13 shows the results of IBFS followed by postprocessing on input I_6.

Table VI shows the average number of sides of modules for each input. These results compare favorably with those in [10], where the average number of sides is between 15.2 and 20 for relatively small benchmarks with only four to eight soft blocks. We also measured the change in wirelength caused by the modest repositioning of the modules resulting from our algorithms. As expected, the wirelength as a result of our algorithm changes very slightly relative to the wirelength before our algorithm was run. The increase in wirelength is always less than 3% and is usually in the neighborhood of 1%–2%. The initial wirelength was estimated by positioning ami33/ami49 blocks so that their centers coincided with the centers of the constraining rectangles. The final wirelength was estimated by repeating the process, but by using the center of the bounding box of the module polygon computed by our algorithm. Code from [19] was used to compute wirelengths.

Our runtime results are displayed in Table VII.

TABLE VII
RUNTIME FOR VARIOUS COMPONENTS OF OUR ALGORITHM ON AMI33 AND AMI49

Code	Run Time (seconds)	
	ami33	ami49
Sweep Plane	5.113	5.337
Min-Cost Max-Flow	0.1	0.11
Post processing	0.446	0.495
Total	5.659	5.843

TABLE VIII
RUNTIMES OF MIN-COST MAX-FLOW ALGORITHM FOR DIFFERENT-SIZED INPUTS ARE SHOWN. THE INPUTS ARE OBTAINED BY MAKING MULTIPLE COPIES OF AMI33 AND PLACING THEM NEXT TO EACH OTHER TO CREATE LARGER FLOORPLANS

number of modules	33	66	132	264
running time	166	431	1674	7284

The worst-case time complexities for the algorithms reported in Section IV is quite high. However, we suggested that the actual time complexities were significantly lower. This is supported by Table VIII. Observe that a doubling in input size results in an increase in run time that is between four- and eightfold, which suggests that the growth rate for runtime is greater than n^2, but less than n^3. Similar runtime results were obtained for GSRC benchmarks consisting of up to 300 modules. For example, the average running time of min-cost max-flow algorithm for n300a is 8.34 s.[2]

VI. Discussion

This paper considered a version of floorplanning under the assumption that approximate relative locations for a floorplan are available. Three algorithms (BFS, IBFS, CBFS) were proposed. Of these, we recommend IBFS as it provides results that can be easily postprocessed to obtain connected modules in a zero-whitespace floorplan (as demonstrated by our experiments). The zero-whitespace assumption places more stress on our algorithms as there is more competition among modules for regions. Recent trends in floorplanning deliberately incorporate whitespace in floorplans to enable subsequent buffer insertion. The algorithms described in this paper can be used when whitespace is present in the floorplan and will, in fact, work better when whitespace is available as there will be less competition among modules for regions. An architect can incorporate whitespace in certain areas of the floorplan by using filler (empty) blocks. Whitespace *within* a module can be modeled by simply using a required area that is larger than the actual area required by the block. Our approach works well for floorplans containing two types of modules, which are: 1) a hard macro with a single fixed shape and 2) a completely soft block. Our approach cannot directly model a hard macro with many possible (fixed) shapes.

Recently, Choi and Bazargan [20] proposed a global floorplacement method to produce good mixed-size placement results by allowing area migration. Our approach may be combined with this approach for mixed-size placements. Also, we believe that more can be done with the postprocessing step to generate more compact-looking modules, once module connectivity has been achieved. However, compact modules can also be obtained by using smaller constraining rectangles. Our experiments used constraining rectangles where the ratio of the area of the constraining rectangle to the area of the module varied from 1.96 to 3.24; i.e., a 96%–224% increase. Smaller constraining rectangles (e.g., where this ratio is 1.21) are likely to overconstrain the problem, resulting in an infeasible problem. We are presently studying iterative methods for selectively reducing the size of the constraining rectangle for some modules. We are also presently studying methods to provide feedback to the designer when the input is not bound-feasible. This will complete all of the steps displayed in Fig. 1.

[2]The LEDA implementation of min-cost max-flow uses capacity scaling and not enhanced capacity scaling, resulting in a higher complexity than the one reported in this paper.

References

[1] A. B. Kahng, "Classical floorplanning harmful?," in *Proc. Int. Symp. Physical Design*, 2000, pp. 207–213.

[2] R. Otten, "Automatic floorplan design," in *Proc. 19th ACM/IEEE Design Automation Conf.*, 1982, pp. 261–267.

[3] D. F. Wong and C. L. Liu, "Floorplan design of VLSI circuits," *Algorithmica*, vol. 4, pp. 263–291, 1989.

[4] H. Murata, F. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle packing by the sequence-pair," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 1518–1524, Dec. 1996.

[5] S. Nakatake, F. Fujiyoshi, H. Murata, and Y. Kajitani, "Module packing based on the BSG-structure and IC layout applications," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 519–530, June 1998.

[6] Y. Pang, C.-K. Cheng, and T. Yoshimura, "An enhanced perturbing algorithm for floorplan design using the O-tree representation," in *Proc. Int. Symp. Physical Design*, 2000, pp. 168–173.

[7] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B^*-trees: A new representation for nonslicing floorplans," in *Proc. Design Automation Conf.*, 2000, pp. 103–110.

[8] K. Sakamushi and Y. Kajitani, "The quarter-sequence (Q-Seq) to represent the floorplan and applications to layout optimization," in *Proc. IEEE Asia Pacific Conf. Circuits Syst*, 2000, pp. 829–832.

[9] X. Hong *et al.*, "Corner block list: An effective and efficient topological representation of nonslicing floorplan," in *Proc. Int. Conf. Computer-Aided Design*, 2000, pp. 8–13.

[10] D. Mehta and N. Sherwani, "On the use of flexible, rectilinear blocks to obtain minimum-area floorplans in mixed block and cell designs," *ACM Trans. Design Automation Electron. Syst.*, vol. 5, pp. 82–97, 2000.

[11] S. Liao, M. Lopez, and D. Mehta, "Constrained polygon transformations for incremental floorplanning," *ACM Trans. Design Automation Electron. Syst.*, vol. 6, pp. 322–342, 2001.

[12] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning through better local search," in *Proc. ACM/IEEE Int. Conf. Computer Design*, 2001, pp. 328–334.

[13] ——, "Consistent placement of macro-blocks using flooplanning and standard-cell placement," in *Proc. Int. Symp. Physical Design*, 2002, pp. 12–17.

[14] X. Tang and D. F. Wong, "Floorplanning with alignment and performance constraints," in *Proc. 39th ACM/IEEE Design Automation Conf.*, 2002, pp. 848–853.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. New York: McGraw-Hill, 2001.

[17] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[18] K. Mehlhorn and S. Naher, *LEDA: A Platform for Combinatorial and Geometric Computing*. New York: Cambridge Univ. Press, 1999.

[19] W. Dai, L. Wu, and S. Zhang. New file formats, benchmarks, results, floorplanner source codes and executables. [Online]. Available: http://www.cse.ucsc.edu/research/surf/GSRC/progress.html

[20] W. Choi and K. Bazargan, "Hierarchical global floorplacement using simulated annealing and network flow area migration," in *Proc. Design, Automation, Test Eur. Conf. Exhibit.*, 2003, pp. 1590–1591.