Constrained “Modern” Floorplanning

Yan Feng  
Department of Mathematical and Computer Science  
Colorado School of Mines  
yfeng@mines.edu

Dinesh P. Mehta*  
Department of Mathematical and Computer Science  
Colorado School of Mines  
dmehta@mines.edu

Hannah Yang  
Strategic CAD Labs, Intel  
2111 NE 25th Ave  
Hillsboro, Oregon  
hyang@ichips.intel.com

ABSTRACT
This paper presents algorithms for a constrained version of the “modern” floorplanning problem proposed by Kalng in “Classical Floorplanning Harmful?” [1]. 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 semi-automated floorplan methodology and (2) in incremental floorplanning. CMFP is shown to be NP 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 (BFS, IBFS, 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 BFS and IBFS are fast and obtain zero whitespace floorplans.

Categories and Subject Descriptors
B.7.2 [Hardware]: Integrated Circuits; F.2.2 [Theory of Computations]: Analysis of Algorithms and Problem Complexity

General Terms
Algorithms

Keywords
Floorplanning, Network flow, Rectilinear Polygons

*This material is based upon work supported by the National Science Foundation under Grant No. CCR-9988338.

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

1. 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, 3, 4, 5, 6, 7, 8, 9].

Kalng [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) Modules shapes should not be restricted to rectangles, L-shapes, and T-shapes. (3) “Round” blocks with 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 Kalng 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 EDA newsgroup:

“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 (Figure 1(a)) and the distance between the farthest pair of points in the module can be large (e.g., module 3 in Figure 1(a)). 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.

**Figure 1:** Floorplans generated by the hybrid MBF-based algorithm (reproduced from [10]).

In this paper, we formulate the problem differently so that the distance between the farthest pair of points in each module is bounded. A max-flow algorithm determines whether the input satisfies feasibility criteria. A min-cost max-flow heuristic is used to compute the floorplan. Our graph-flow based solutions take a global and integrated view that simultaneously includes all modules and is an improvement over the previous approach which floorplanned modules sequentially. Zero whitespace solutions are guaranteed. Experimental results show that module connectivity is achieved and that the modules are compact and have few sides.

The fixed die aspects of Kahan's formulation have also been addressed recently by Adya and Markov in [12, 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.

Figure 2 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 post-processing 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 post-processing 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.)

Section 2 defines CMFP. Section 3 describes an algorithm that determines bound-feasibility. Section 4 describes algorithms that assign regions to modules. Section 5 discusses the postprocessing step and provides
2. CMFP: THE CONSTRAINED MODERN FLOORPLANNING PROBLEM

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 $(x_i, y_i, r_x^i, r_y^i, A_i)$. 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 to be assigned to module $m_i$. $r_x^i$ and $r_y^i$ 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) Area assigned to $m_i$ must be inside the fixed die and should be within the constraining rectangle bounded by $[x_i - r_x^i, x_i + r_x^i]$ in the $x$-direction and $[y_i - r_y^i, y_i + r_y^i]$ in the $y$-direction. (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 $r_x^i$ and $r_y^i$ to be half the width and height, respectively, of the module, and setting $A_i = 4r_x^i r_y^i$.

Theorem 1. CMFP is NP-hard (proof omitted).

3. FLOW-BASED BOUND-FEASIBILITY COMPUTATION

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

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

(1)

Figure 3 shows an example of an infeasible input. (See Figure 10 for another example.)

![Figure 3: Two constraining rectangles $A$ and $B$ each have 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.](image)

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 $Q$. Figure 4 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 $E = E_1 \cup E_2 \cup E_3$.

![Figure 4: Five rectangles $A, B, C, D, E$ 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, 3, 4, 7, 8, 9, 10, and 11; $C$ is a red rectangle consisting of 4, 5, 11, and 12; $D$ is the blue rectangle and consists of regions 6, 7, 8, 9, 13, and 14; $E$ is the green rectangle consisting of 9, 10, 11, 12, 14, and 15. Collinear edges have been staggered for clarity.](image)
flow conservation at \( v \). So \( \sum_{v \in R} f(u, v) \leq A_u \) and \( \sum_{u \in R} f(u, v) \leq \text{Area}(v) \). 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 are either the four vertices of each input rectangle and the points of intersection between a pair of orthogonal sides (See Figure 4). Note that the number of intersections is bounded by \( 4n^2 \) as there are \( 2n \) vertical sides and \( 2n \) horizontal sides. So, \( r \) 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, \( |R| = f = O(n^2) \).

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

**Proof.** An \( O(n^2) \) planescape algorithm is used to identify the regions of the dissection. The maximum flow algorithm runs in \( O(|V||E| \log |E|) \) time [16]. In the worst case, \( |V| = O(n^2) \) and \( |E| = O(n^3) \), resulting in a complexity of \( O(n^4 \log n) \). In practice, for tighter constraining rectangles, we expect the number of regions to be \( \Theta(n) \), giving \( |V| = \Theta(n) \). We also expect \( |E| = \Theta(n) \) because a region will typically belong to a constant number of modules. Under these assumptions, the runtime is \( O(n^2 \log n) \).

### 4. 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 several disconnected components, thus violating 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 costs and edge flows while maintaining maximum flow through the network. The difference is that we have to assign costs to edges in a way that forces the assignment of connected regions to soft blocks.

#### 4.1 BFS Algorithm

In the BFS algorithm, we perform a breadth first search on a graph corresponding to the regions of each constraining rectangle \( i \) (each region is a vertex; vertices representing adjacent regions are joined by an edge) starting at a region near the center. The breadth first number [16] computed for each region \( r \) is the cost of the edge in the flow graph from \( i \) to \( r \). Figure 6 shows the breadth first numbering of regions in a constraining rectangle.

**Figure 6:** Breadth first numbering of regions belonging to a constraining rectangle starting by labeling the region containing the center point 0.

The philosophy of this approach is that regions will be “filled” in increasing order of cost by the min-cost algorithm, resulting in connected blocks. However, there are some scenarios such as in Figure 7 where the algorithm cannot find connected solutions, even though they may exist.

#### 4.2 Improved BFS Algorithm

In the improved-BFS (IBFS) algorithm, we address the situation when edges \( (l_i, r) \) and \( (l_j, r) \) have the same cost 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 Figure 4. Their constraining rectangles have areas 10,000 and 14,300, respectively. \( A \) must be assigned 8000 units and \( B \) must be assigned 12,000 units. Consider region 2 which lies in \( A \cap B \) with area 2700. In BFS, edges \((A, 2)\) and
Figure 7: An 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 5 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.

(4.2) would be assigned a cost of 1 unit. In IBFS, we take into account that A has exclusive access to the 6,300 units of region 1 and B has exclusive access to the 5,100 units of region 3. Thus, A needs an additional 3,700 units from shared regions and B needs an additional 7,200 units. These quantities are used to assign a cost of $1 + \frac{3,700}{6,300}$ to edge (A, 2) and a cost of $1 + \frac{7,200}{5,100}$ 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.

### 4.3 Compromise BFS Algorithm

In the compromise-BFS algorithm (CBFS), if a region $r$ is shared by modules $m_1...m_k$, we replace the vertex corresponding to region $r$ in the flow graph with vertices $v_1...v_k$. An edge is created between each module in $m_i$, $1 \leq i \leq k$ and every region $v_i$, $1 \leq i \leq k$. The edge between $m_i$ and $v_i$ is assigned its original cost from BFS. All the other edges have cost $c$, which is a large constant. The idea here is that each module has priority to a portion of each region that it shares.

Figure 8 shows an example involving modules B, C, and E and regions 4, 5, 11, and 12 from Figure 4.

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

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

### 5. POSTPROCESSING STEP AND EXPERIMENTAL RESULTS

#### 5.1 Postprocessing Step

Although the three BFS-based algorithms specify how much area of a region should be assigned to a module, they don’t 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 post-processing steps outlined in this section are concerned with making modules connected and assigning an exact geometric area to each module.

A region graph is an adjacency graph based on the regions of the floorplan. Each region is a vertex in this graph. Two vertices are joined by an edge if and only if the corresponding regions share a common segment on their boundaries. The output of any of the BFS algorithms can be represented in the region graph $RG$ as follows: each region vertex contains a list of the modules that were allocated 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 $R_m$.

---

*(Note: The text is cut off at this point, and additional content is not visible.)*
\( m \) 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:

**Lemma 2.** 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 very few small connected components (3) the connected areas are near each other and are separated by at most one intervening region. Thus our postprocessing step simply exchanges module-assignments among neighboring regions to make the output graph-connected.

**Lemma 3.** A graph-connected module-region assignment does not imply the existence of a connected floorplan (see Figure 9).

![Figure 9: The figure shows 5 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.](image)

Our geometric-connectivity post-processing step only considers regions that will be shared by several modules and floorplans by regions such as 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.

### 5.2 Experiments

The planesweep algorithm that identifies regions, their areas, and adjacencies between regions was implemented in Java. The infeasibility-detection algorithm of Section 3 and the min-cost max-flow algorithm of Section 4 were implemented on a Linux workstation running on a Pentium processor in C++ using LEDA [17].

Figure 10 shows the output of our infeasibility-detection algorithm when it was run on ami33.

![Figure 10: A 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.](image)

Table 1 describes the benchmarks used to test our algorithms. Tables 2 and 3 present metrics comparing our algorithms for assigning regions to modules. These metrics determine whether the algorithms produce an output that will be amenable to post-processing. Algorithms that do not perform well with respect to these metrics were discarded. Thus, MaxFlow was discarded after Table 2 because of the large number of disconnects. CBFS was discarded after Table 3 because of the large number of modules sharing a region.

Tables 4 and 5 display the same metrics as Table 2 and 3, 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 towards 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. We found that this often improved the solution quality as shown in the tables.

Next, the post-processing 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. Figure 11 shows the results of IBFS followed by post-processing on input \( k_6 \).

Our timing results are displayed below in Table 6.

### 6. Discussion

This paper considered the modern version of the floorplanning under the assumption that approximate relative locations for a floorplan are available. Three al-
Table 1: 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.

<table>
<thead>
<tr>
<th>Input ID</th>
<th>( I_1 )</th>
<th>( I_2 )</th>
<th>( I_3 )</th>
<th>( I_4 )</th>
<th>( I_5 )</th>
<th>( I_6 )</th>
<th>( I_7 )</th>
<th>( I_8 )</th>
<th>( I_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark</td>
<td>ami33</td>
<td>ami33</td>
<td>ami33</td>
<td>ami33</td>
<td>ami33</td>
<td>ami33</td>
<td>ami49</td>
<td>ami49</td>
<td>ami49</td>
</tr>
<tr>
<td>CR ratio</td>
<td>1.4</td>
<td>1.6</td>
<td>1.6</td>
<td>1.7</td>
<td>1.6</td>
<td>1.8</td>
<td>1.4</td>
<td>1.6</td>
<td>1.7</td>
</tr>
<tr>
<td>No of regions</td>
<td>285</td>
<td>335</td>
<td>335</td>
<td>344</td>
<td>346</td>
<td>409</td>
<td>508</td>
<td>542</td>
<td>564</td>
</tr>
</tbody>
</table>

Table 2: 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 MaxFlow.

<table>
<thead>
<tr>
<th>Input ID</th>
<th>( I_1 )</th>
<th>( I_2 )</th>
<th>( I_3 )</th>
<th>( I_4 )</th>
<th>( I_5 )</th>
<th>( I_6 )</th>
<th>( I_7 )</th>
<th>( I_8 )</th>
<th>( I_9 )</th>
<th>( I_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>3(8)</td>
<td>3(5)</td>
<td>3(2)</td>
<td>3(4)</td>
<td>3(9)</td>
<td>3(1)</td>
<td>3(4)</td>
<td>3(7)</td>
<td>4(1)</td>
<td>3(7)</td>
</tr>
<tr>
<td>IBFS</td>
<td>3(3)</td>
<td>3(1)</td>
<td>3(2)</td>
<td>3(2)</td>
<td>3(1)</td>
<td>3(4)</td>
<td>3(3)</td>
<td>3(2)</td>
<td>3(2)</td>
<td>3(4)</td>
</tr>
<tr>
<td>CBFS</td>
<td>7(4)</td>
<td>8(1)</td>
<td>8(3)</td>
<td>8(3)</td>
<td>8(4)</td>
<td>8(5)</td>
<td>8(2)</td>
<td>8(1)</td>
<td>8(3)</td>
<td>8(3)</td>
</tr>
</tbody>
</table>

Table 3: 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 assigned 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 8 regions that were assigned 3 modules and that no regions were assigned 4 or more modules. Observe that the numbers for BFS and IBFS are superior to those for CBFS.

<table>
<thead>
<tr>
<th>Input ID</th>
<th>( I_1 )</th>
<th>( I_2 )</th>
<th>( I_3 )</th>
<th>( I_4 )</th>
<th>( I_5 )</th>
<th>( I_6 )</th>
<th>( I_7 )</th>
<th>( I_8 )</th>
<th>( I_9 )</th>
<th>( I_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Center Adjustment</td>
<td>7</td>
<td>9</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>After Center Adjustment</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4: Number of disconnected regions before and after center adjustment for IBFS.

<table>
<thead>
<tr>
<th>Input ID</th>
<th>( I_1 )</th>
<th>( I_2 )</th>
<th>( I_3 )</th>
<th>( I_4 )</th>
<th>( I_5 )</th>
<th>( I_6 )</th>
<th>( I_7 )</th>
<th>( I_8 )</th>
<th>( I_9 )</th>
<th>( I_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Center Adjustment</td>
<td>3(3)</td>
<td>3(1)</td>
<td>3(2)</td>
<td>3(2)</td>
<td>3(1)</td>
<td>3(1)</td>
<td>3(3)</td>
<td>3(3)</td>
<td>3(3)</td>
<td>3(1)</td>
</tr>
<tr>
<td>After Center Adjustment</td>
<td>3(1)</td>
<td>3(2)</td>
<td>3(3)</td>
<td>3(1)</td>
<td>3(4)</td>
<td>3(1)</td>
<td>3(2)</td>
<td>3(2)</td>
<td>3(2)</td>
<td>3(4)</td>
</tr>
</tbody>
</table>

Table 5: 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 3 or more modules. The * indicates that the number of regions with two modules is not of interest.
7. REFERENCES


