NETWORK FLOW BASED MULTI-WAY PARTITIONING WITH AREA AND PIN CONSTRAINTS *

Huiqun Liu and D. F. Wong
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

ABSTRACT

Network flow is an excellent approach to finding min-cuts because of the celebrated max-flow min-cut theorem. However, for a long time, it was perceived as computationally expensive and deemed impractical for circuit partitioning. Only until recently, FBB [1] successfully applied network flow to two-way balanced partitioning and for the first time demonstrated that network flow was a viable approach to circuit partitioning. In this paper, we present FBB-MW, which is an extension of FBB, to solve the problem of multi-way partitioning with area and pin constraints. Experimental results show that FBB-MW outperforms the FM-based MW-part program in the TAPIR package[10].

1. INTRODUCTION

Circuit partitioning is crucial in VLSI system design. Multi-way partitioning is becoming very important with the ever increasing system size. A target device such as an FPGA usually has an upper bound for both area and I/O pins. For multiple-FPGA system design, the objective for circuit partitioning is to minimize the total number of crossing nets while satisfying area and pin constraints. Traditional multi-way partitioning algorithms which only minimize the total cut nets are no longer applicable, and hence recently several algorithms [3,4,5,6,7,8] have been proposed for multi-way partitioning with area and pin constraints.

Network flow is an excellent approach to finding min-cuts because of the celebrated max-flow min-cut theorem [9]. However, for a long time, it was perceived as computationally expensive and deemed impractical for circuit partitioning. Only until recently, FBB [1] successfully applied network flow to two-way balanced partitioning and for the first time demonstrated that network flow was a viable approach to circuit partitioning. Later, [2] improved FBB by introducing node-selection heuristics based on linear placement of the nodes.

In this paper we present FBB-MW, which is an extension of FBB, to solve the problem of multi-way partitioning with area and pin constraints. We first give an improvement of FBB by finding the "most desirable" min-cut during every iteration of FBB. This is based on the observation that there are usually many min-cuts after a maximum-flow computation. Using the "most desirable" min-cut reduces the number of iterations in FBB which would in turn reduce total runtime and final cut-size. In FBB-MW, we apply the techniques in FBB and its improvements to multi-way partitioning. While FBB only minimizes the number of cut nets without taking into consideration of the total number of pins for each partitioned component, in order to satisfy the area and pin limits, we must consider both the primary I/O nodes and the interconnecting nets which will occupy the I/O pins. By a suitable network modeling of the I/O nodes, we can minimize the total number of pins for one component by maximum flow computation. Experimental results show that FBB-MW outperforms the FM-based MW-part program in the TAPIR package [10].

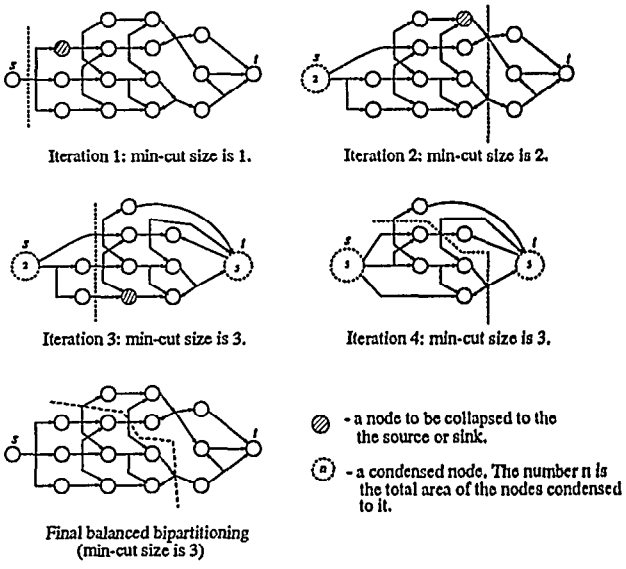


Figure 1. *Using FBB for balanced bipartition*

2. IMPROVEMENT OF FBB

2.1. Overview of FBB

Network flow based partitioning method was once overlooked to be a practical partitioning method because of its relatively high complexity. Recently, [1] proposed the FBB algorithm for flow based balanced circuit bipartitioning. By proper net-modeling and employing incremental flow computation, FBB not only yields better partitioning results, but also is efficient in computation time. [2] further improved FBB by introducing node selection heuristics based on linear placement of the nodes.

*This work was partially supported by a grant from the Avant! Corporation.

ISPD '97 Napa Valley, California USA

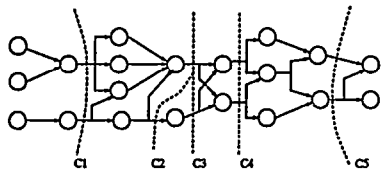


Figure 2. *There are multiple min-cuts in the flow network partitioning the network into subsets of different area.*

FBB applies an efficient max-flow min-cut heuristic to repeatedly cut the network to meet the area limit. The repeated max-flow min-cut process was implemented efficiently by using incremental flow computation. It is not necessary to do the max-flow computation from scratch in each iteration, only additional flow is added to saturate the bridging edges from iteration to iteration. It was proved in [1] that the repeated cut process takes the same asymptotic time complexity as that of one max-flow computation.

Figure 1 shows an example of using FBB for balanced bipartition. For simplicity, the net modeling is not shown in the figure. In each iteration, max-flow is computed and a min-cut is found. Then all the nodes in the smaller side of the min-cut are condensed to form one seed node and a new node is collapsed to this seed node, so that more flows can be pushed through the network. This process goes on until a balanced partition is found.

2.2. Improvement of FBB

2.2.1. Finding the most desirable min-cut

In each iteration of FBB, after obtaining the max-flow, FBB used $X_s = \{v | \exists$ an augmenting path from s to $v\}$ as the min-cut. An augmenting path from v to u is a path that more flows can be pushed through it. An important observation is that there usually exist more min-cuts in the flow network (as shown in Figure 2). Besides X_s, X_t defines a min-cut that is closest to the sink where $X_t = \{v | \exists$ an augmenting path from v to $t\}$ and there may exist more min-cuts in between the two. It is easy to show that for any min-cut $(X, \overline{X})$, $X_s \subseteq X \subseteq (V - X_t)$.

One improvement to FBB is that in each iteration after the max-flow computation, we try to find the min-cut that cuts the network into subsets with area as close to the area limit as possible. We observe that when collapsing a node to the source (or sink) and then pushing additional flow, the min-cut size is monotonically increasing. By first exploring the existing min-cuts and finding one closest to the area limit, we obtain a subset with a larger area without increasing the min-cut size.

A min-cut partitions a flow network with total area A into two parts: X and $\overline{X}$, where the source $s \in X$ and the sink $t \in \overline{X}$. Let $\tilde{A}$ be the area constraint (*i.e.* $\tilde{A}$= $0.5A$ for balanced bipartitioning). Let $\delta = min(|\tilde{A} - area(X)|, |\tilde{A} - area(\overline{X})|)$ for a min-cut $(X, \overline{X})$. The value δ measures the deviation of the partition from the specified area limit. Among all the min-cuts in the flow network, the one with minimum δ is called *the most desirable min-cut*, which is a min-cut closest to the area limit $\tilde{A}$.

In Figure 2, min-cut C_1 corresponds to X_s and C_5 corresponds to X_t. C_2, C_3, C_4 are other min-cuts in the flow network. If the area limit $\tilde{A}$=10, then C_2 would be the most desirable min-cut. If $\tilde{A}$=8, then C_4 which partitions the network into subsets of area 13 and 7 would be a min-cut that is closest to the area limit.

After obtaining max-flow in the network, all the existing min-cuts partition the flow network G into non-overlapping

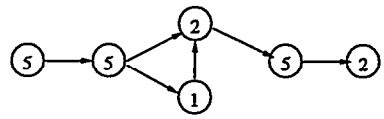


Figure 3. *The min-cut graph H corresponding to the network in Figure 2. The number inside each node is the total area of the subset represented by the node.*

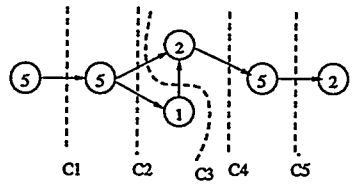


Figure 4. *The set of unidirectional cuts in the min-cut graph H*

subsets. We define the *min-cut graph* H from G as follows:

1. Each subset separated by the min-cuts in G is represented by a node in H.
2. If all edges from subset s_1 to s_2 in G are saturated and all edges from subset s_2 to s_1 in G have zero flow, then add an edge from node s_1 to node s_2 in H.
3. Each node s in H is associated with an area which is equal to the total area of all the nodes in the subset represented by s.

Figure 3 is the corresponding min-cut graph H of the network in Figure 2. The five min-cuts in G (Figure 2) partition the network into six subsets, each of which is represented by a node in H (Figure 3).

Lemma 1: *The min-cut graph H is a directed acyclic graph.*

A *unidirectional cut* of H is defined as a cut that partitions H into $(Y, \overline{Y})$, such that all the cut edges are from nodes in Y to nodes in $\overline{Y}$. Note that the set of unidirectional cuts in Figure 4 corresponds to the set of min-cuts in Figure 2. This is true in general and we have the following lemma.

Lemma 2: *There is a 1-1 correspondence between the set of all unidirectional cuts in H and the set of all min-cuts in G.*

Due to space limitation, the proofs of Lemma 1 and 2 are omitted here. To find a most desirable min-cut that is closest to the area limit, we can first construct the min-cut graph H and then find a unidirectional cut which partitions the network into subsets with a desirable area. Since constructing the exact H can be time consuming, we used a greedy heuristic **DMC** to build a min-cut graph and find a unidirectional cut with area close to the area constraint.

Procedure DMC: finding a desirable min-cut

1. Search from the sink t;
   $X_0 = \{v | \exists$ an augmenting path from v to $t\}$;
2. Search from the source s;
   $X_1 = \{v | \exists$ an augmenting path from s to $v\}$;
   $i = 2$;
3. Select an unmarked node v incident to $X_j (1 \leq j < i)$;
   $X_i = \{u | \exists$ an augmenting path from v to u and u is unmarked$\}$;
4. $i \leftarrow i + 1$;
   If all nodes are marked, then goto step 5; else goto step 3;
5. Select k with $\sum_{i=1}^{k} area(X_i) \leq \tilde{A}$ and $max(\sum_{i=1}^{k} area(X_i))$;
   let $X = \cup_{i=1}^{k} X_i$;
   return X;

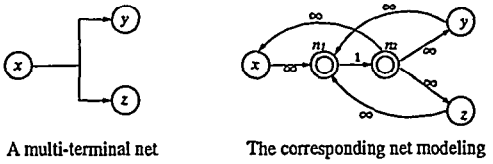


Figure 5. *Net modeling*

From steps 1 to 4 a min-cut graph is constructed, but it is not the exact H as defined above since some nodes in H may be merged into one node here. A simple strategy is given in step 5 to find a min-cut that has area close to the area limit. It can be easily proved that $X = \cup_{i=1}^{k} X_i$ is a unidirectional cut in the min-cut graph and thus a min-cut in the flow network. More intelligent strategies can be used in step 5 for finding a desirable min-cut by searching the min-cut graph. In addition to checking the total area, the total number of pins for $X = \cup_{i=1}^{k} X_i$ can also be calculated and compared with the pin limit.

When the size of the network is much larger than the area limit, we do not need to construct H on the whole network. Instead, we can build a partial min-cut graph as follows. In step 5, if $\sum_i area(X_i)$ exceeds a certain limit, we leave the rest of the nodes to be in one subset of the min-cut graph.

After a max-flow computation on the flow network, procedure **DMC** can be used to find a desirable min-cut. Note that when the min-cut found does not meet the area limit, incremental max-flow computation will be performed again and another desirable min-cut is found by procedure **DMC**. With this process going on, the min-cut approaches the area limit in each iteration.

2.2.2. Net Modeling

To make network flow based method suitable for circuit partitioning, [1] gave a net modeling of the hyperedges so that by max-flow min-cut computation, the min-cut found is the total number of cut nets. Here we make a simplification in the modeling of two-terminal nets. We construct a flow network $G' = (V', N')$ from $G = (V, N)$ as follows (see Figure 5):

1. V' contains all the nodes in V.
2. For a multi-terminal net $n = (v_1, v_2, ...v_k)$, add two nodes n_1, n_2 in V' and add a bridging edge (n_1, n_2) in N' with capacity 1. For each node $v \in \{v_1, v_2, ..., v_k\}$ incident on net n, add two edges (v, n_1) and (n_2, v) in N', with capacity ∞ on these edges.
3. For a two-terminal net $n = (u, v)$, add two edges (v, u) and (u, v) with each having capacity 1.
4. For each node $v \in V$, its corresponding node in V' has the same area as in V. For a node in V' but not in V, assign area as 0.

Here we distinguish between a two-terminal net and a multi-terminal net, so that we do not need to add the extra two nodes and the bridging edge for a two-terminal net. This can reduce the size of the resulting network and thus speed up the max-flow computation. Our multi-way partitioning algorithms in Section 3 are based on the above net modeling. The following lemma says that the net modeling is correct and its proof is similar to the one in [1].

Lemma 3: *Let $(X', \overline{X'})$ be a min-cut of capacity C in G', and $(X, \overline{X})$ be the corresponding cut in G, we have $(X, \overline{X})$ is a minimum net cut in G and the number of cut nets is equal to C.*

3. MULTI-WAY PARTITIONING

In this section, we introduce algorithm FBB-MW, an extension of FBB to multi-way circuit partitioning with area and pin constraints. Algorithms which only minimize the total number of interconnections will not be useful for solving this problem, since they can not guarantee that each component can meet the pin limit as the cut nets may be distributed unevenly among the components even if the total is minimized. Besides the crossing nets, the primary I/O nodes will also occupy the I/O pins and should be taken into consideration. Moreover, it is desirable to find a partition that uses as few components as possible in order to reduce the total cost of the design. In this section, we will first give the problem formulation and then present our network flow based algorithms.

3.1. Problem Formulation

For a circuit $G = (V, N)$, V is a set of nodes with each node associated with an area, N is a set of nets where a net is a subset of V. Given the upper bound for both the area ($\tilde{A}$) and the number of pins ($\tilde{P}$), the multi-way partitioning problem is to partition V into k non-overlapping subsets $V_1, V_2, ...V_k$, such that (1) $V = \cup_{i=1}^{k} V_i$; (2) $area(V_i) \leq \tilde{A}$ for $i = 1, ..., k$ and (3) $pin(V_i) \leq \tilde{P}$ for $i = 1, ..., k$, with the objective of minimizing k and $\sum_{i=1}^{k} pin(V_i)$.

Each subset is also called a component. Here $area(V_i) = \sum_{v \in V_i} area(v)$ and $pin(V_i)$ is the total number of pins for component V_i. The objective is to minimize the number of components and to minimize the total number of pins while each component must satisfy the specified area and pin constraints. Notice that the total pins for one component is comprised of pins for the interconnecting nets among the components and the primary I/Os.

3.2. Algorithm 1

One direct extension of FBB to multi-way partitioning is to iteratively apply the max-flow min-cut process to find one component at a time that meets the area and pin constraint, until every node in G' is assigned to a component. A *feasible component* is a subset of V which satisfy the area and pin limit. **FC** is a heuristic for finding one feasible component with area as large as possible.

Procedure FC: finding a feasible component.
1. Pick a source s and a sink t; $F \leftarrow \phi$;
2. Compute max-flow in the flow network;
3. Call procedure **DMC** to find a desirable min-cut $C = (X, \overline{X})$;
4. If $C \geq \tilde{P}$, then return(F);
   else $assign(X, F)$; $assign(\overline{X}, F)$;
5. If $\lambda\tilde{A} \leq area(X) \leq \tilde{A}$ then
   return(F);
   else if $area(X) < \tilde{A}$, then
   collapse nodes in X to s;
   collapse to s a node $v \in \overline{X}$ incident on s;
   else if $area(X) > \tilde{A}$, then
   collapse nodes in $\overline{X}$ to t;
   collapse to t a node $v \in X$ incident on t;
6. goto step 2;

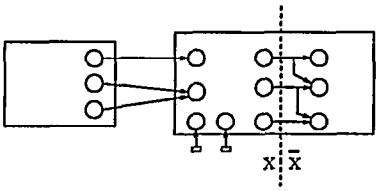


Figure 6. *The total pins for component X consist of three parts: PI/PO, cut nets to $\overline{X}$, cut nets to other components*

In step 2, the max-flow in the flow network is computed. Incremental flow computation is employed here, as only additional flow is added to saturate the edges from iteration to iteration. Procedure **DMC** is called in step 3 to find a desirable min-cut. In step 4, F is used to save the best feasible subset that has been found so far. In function $assign(X, F)$, if $pin(X) \leq \bar{P}$ and $area(F) < area(X) \leq \bar{A}$, then assign X to F since X is a larger feasible subset than F. The min-cut calculated in step 3 is the number of cut nets between X and $\overline{X}$, not including the primary I/Os and interconnecting nets to earlier partitioned components. So in function $assign(X, F)$, we have to count the number of these type of nodes and add it to the min-cut size to get the total number of pins. In step 5, if the area of X is within range of $\lambda\bar{A}$ to $\bar{A}$ ($0 < \lambda \leq 1$, *i.e.* $\lambda = 80\%$), then procedure **FC** terminates and returns F, else nodes are condensed to one seed node. Then control goes back to step 2 again to find the next desirable min-cut by pushing more flow in the network.

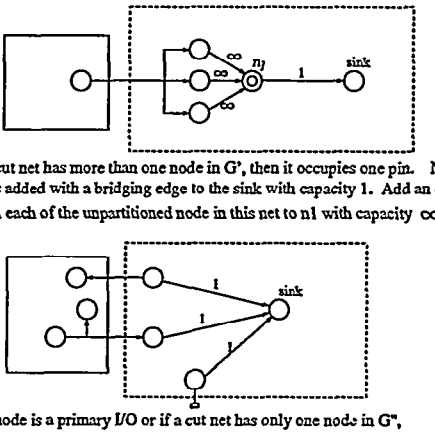
Similar to the proof in [1], we can show that procedure **FC** takes time $O(|V||E|)$ for a network $G' = (V, E)$. Each augmenting path computation takes time $O(|E|)$ and by incremental flow computation, the total number of augmenting path found is $O(|V|)$. Therefore the time complexity of finding a feasible component is $O(|V||E|)$.

Algorithm 1 is designed for multi-way partitioning. It repeatedly calls procedure **FC** to find one feasible component at a time. After one component is found, the flow on the edges in the rest of the network is reset to zero before finding the next component. The time complexity of algorithm 1 is $O(k|V||E|)$ with k being the number of partitioned components.

3.3. Algorithm 2

In procedure **FC** of algorithm 1, after the max-flow computation in each iteration, the min-cut obtained measures the number of cut nets rather than the total number of pins. Hence in order to meet the pin constraint, the number of I/Os is counted and added together with the min-cut to get the total number of pins. The disadvantage is that during the max-flow computation, there is little control on how many I/Os will be included in X or $\overline{X}$. The random distribution of these I/Os sometimes results in being unable to find a large feasible subset while the min-cut size is still relatively small. Therefore it is important to model the I/O nodes properly.

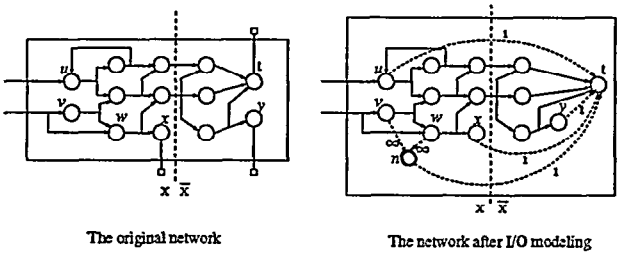
Before we partition a network G', it already has some I/Os which come from two sources: (1) A primary I/O node; (2) A cut net with some nodes in G' and some nodes in other previously partitioned components. We refer to these two type of nodes as I/O nodes in G'. When we partition G' into $(X, \overline{X})$, the total number of pins for subset X includes two parts: the number of cut nets to $\overline{X}$ and the number of I/O nodes. In the discussion below, we use the following notations. $pin(X)$ denotes the total number of pins for a subset X, $net(X, \overline{X})$ is the number of crossing nets from X to $\overline{X}$ and $io(X)$ is the number of I/Os for subset X.



(a) If a cut net has more than one node in G', then it occupies one pin. Node n1 is added with a bridging edge to the sink with capacity 1. Add an edge from each of the unpartitioned node in this net to n1 with capacity ∞ .

(b) If a node is a primary I/O or if a cut net has only one node in G', then add a bridging edge from this node to the sink with capacity 1.

Figure 7. *Modeling of the I/O nodes*



The original network

The network after I/O modeling

Figure 8. *Example of a network before and after I/O modeling*

As shown in Figure 6, component X has eight pins: three cut nets to $\overline{X}$ and five I/Os (which consist of two primary I/Os and three cut nets from a previously partitioned component). We model all the I/O nodes in G' to construct G'' as follows:

1. All nodes and edges in G' are in G''.
2. For a cut net with more than one node in G', add a virtual node n_1. Add an edge from each unassigned node in the net to n_1 with capacity ∞, then add a bridging edge from n_1 to the sink with capacity 1 (Figure 7(a)).
3. If a node v is a primary I/O node, then add a bridging edge from v to the sink, with capacity 1. For a cut net with only one node v in G', add a bridging edge from v to the sink with capacity 1 (Figure 7(b)).

Figure 8 shows an example of a flow network after I/O modeling. With the I/O modeling, we can derive good properties as stated in Lemmas 4 and 5.

Lemma 4: *For a min-cut $(X, \overline{X})$ in G'', the cut size C is equal to the total number of pins for X.*

Proof: For any edge that is cut by the min-cut, it is either a cut net or a bridging edge to the sink for I/O modeling. As the capacity on such an edge is 1, it is counted exactly once in the min-cut. We have $C \leq net(X, \overline{X}) + io(X)$. On the other hand, if a net is cut, then it is counted as one in the min-cut. If an I/O node is in X, then any bridging edge from this node to the sink must be cut and counted once in the min-cut. Therefore $net(X, \overline{X}) + io(X) \leq C$. From the above analysis, we have $C = net(X, \overline{X}) + io(X)$. Since $pin(X) = net(X, \overline{X}) + io(X)$, this leads to $C = pin(X)$. Therefore, the min-cut size C is equal to the total number of pins for X. □

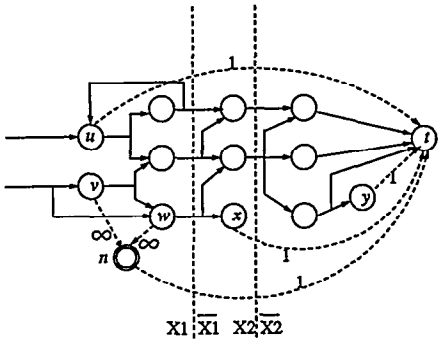


Figure 9. *Comparison of two min-cuts*

As demonstrated in Figure 8, before the I/O modeling, the min-cut size for $(X, \overline{X})$ is two. The total pins for X should be five because two additional I/O pins are used for the two cut nets to other partitioned components and one I/O pin is occupied by the primary output node x. After the I/O modeling, the min-cut size for $(X, \overline{X})$ is five which is equal to the total number of pins for X.

Lemma 5 compares two min-cuts that cut the network into different area and validates the benefit of using procedure **DMC** to find a desirable min-cut that has a large area as close to the area limit as possible.

Lemma 5: *If $(X_1, \overline{X_1})$ and $(X_2, \overline{X_2})$ are two min-cuts with the same cut size in G'' such that $X_1 \subseteq X_2$, then $pin(X_1) = pin(X_2)$, $net(X_1, \overline{X_1}) \geq net(X_2, \overline{X_2})$ and $pin(\overline{X_1}) \geq pin(\overline{X_2})$.*

Proof: If $(X_1, \overline{X_1})$ and $(X_2, \overline{X_2})$ are two min-cuts with the same cut size, then by Lemma 4, $pin(X_1) = pin(X_2)$. As $X_1 \subseteq X_2$, so $io(X_1) \leq io(X_2)$ and $io(\overline{X_1}) \geq io(\overline{X_2})$. It is true that $pin(X) = net(X, \overline{X}) + io(X)$ for $X = X_1, X_2$, thus $net(X_1, \overline{X_1}) \geq net(X_2, \overline{X_2})$. Further, we have $net(X_1, \overline{X_1}) + io(\overline{X_1}) \geq net(X_2, \overline{X_2}) + io(\overline{X_2})$, this leads to $pin(\overline{X_1}) \geq pin(\overline{X_2})$. □

By Lemma 5, for two min-cuts, the one with a larger area is better because it not only has an area closer to the area limit, but also results in fewer number of cut nets and fewer occupied I/O pins for the rest of the network to be partitioned later. Figure 9 shows an example. With $X_1 \subseteq X_2$, the two min-cuts has the same cut size which means X_1 and X_2 has the same number of pins, yet X_1 has three cut nets to $\overline{X_1}$ and X_2 has only two cut nets to $\overline{X_2}$. $\overline{X_2}$ has four I/O pins in total and $\overline{X_1}$ has six I/O pins. Therefore, $\overline{X_2}$ has a smaller area and fewer number of pins than $\overline{X_1}$.

We designed algorithm 2 for multi-way partitioning with I/O modeling. Similar to algorithm 1, it iteratively calls procedure **FC** to find one feasible component at a time. But procedure **FC** is modified as follows: after selecting the source and sink, G'' is constructed from G' by the I/O modeling process. Then max-flow computation is repeatedly applied on G'' to find a min-cut until either the area or the pin constraint is met. By Lemma 4, the min-cut obtained by procedure **DMC** in each iteration is the total number of pins for X with the source $s \in X$. By Lemma 5, procedure **DMC** picks a better min-cut which has fewer cut nets and occupies fewer I/Os in the rest of the network.

3.4. Algorithm FBB-MW: the Merging of Algorithms 1 and 2

By the net modeling of the I/O nodes, Algorithm 2 tries to minimize the total number of pins for each component. Yet one shortcoming is that by adding extra bridging edges, more flows can be pushed in the network, which tends to increase the number of cut nets. To solve this, we designed the third algorithm FBB-MW for flow-based multi-way partitioning which is a combination of algorithm 1 and 2.

Two stages are involved to find one feasible component. In the first stage, when the number of I/O nodes are not significant for the min-cut, we use algorithm 1 to repeatedly cut the network to get min-cuts, which measures the number of cut-nets. In the second stage, when the partitioning result is more sensitive to the distribution of the I/O nodes, we switch from algorithm 1 to algorithm 2. The feasible subset found by algorithm 1 are condensed to be the source node and all the I/O nodes in the network are modeled to form G''. Repeated max-flow computation is then applied to find min-cut, which is equal to the total number of pins for the component.

As algorithm 1 minimizes the number of cut nets but has no control on the distribution of I/O nodes, algorithm 2 is further used to control the number of I/O nodes in component X wich contains the source of the network. The I/O modeling guarantees that the total number of pins is minimized while satisfying the constraints. Experiments shows that FBB-MW produces better results than algorithm 1 and algorithm 2, yet FBB-MW does not increase time complexity. FBB-MW has time complexity $O(k|V||E|)$, which is the same as algorithm 1 and 2.

After we obtain the multi-way partitioning with algorithm FBB-MW, postprocessing is performed to further improve the result. We do a pairwise merge to remove small components and to reduce the number of components. One possible improvement to further reduce the number of pins is to first merge two components, then repartition the merged subsets into two components. Replication algorithms can also be applied to further reduce cut nets among the components and the total number of pins.

4. EXPERIMENTAL RESULTS

We implemented FBB-MW algorithm in C language on IBM RS6000 workstation and tested on the circuits from MCNC Partition93 Benchmark. Table 1 shows the size of the circuits we used for the experiments. For circuits of different sizes, we tried different area limit and pin limit. In Table 2, we compare our partitioning result in FBB-MW with the MW-part of TAPIR package[10], which employs FM based algorithm for multi-way partitioning under predefined area and pin constraints. Compared with TAPIR, FBB-MW gets better results in terms of the number of components, the total number of pins and cut nets.

As shown in Table 2, FBB-MW results in fewer number of components than TAPIR under the same area and pin constraint. For some of the circuits such as C6288, FBB-MW only results in two components while TAPIR gets more. FBB-MW also results in fewer number of total pins and cut nets. For circuit s38417 under area limit 5000 and pin limit 200, FBB-MW results in more number of pins than the FM-base method. This is because FBB-MW partitioned the circuit into 6 components and the nodes are more densely packed, while the FM-based method partitioned it into 9 components. However, FBB-MW still gets fewer number of cut nets in this case. For most of the other experiments, FBB-MW not only yields fewer number of components, but also fewer number of pins and cut nets.

Our efficient implementation of FBB-MW enables it to partition large benchmark circuits with reasonable running time. For circuits such as C5315, C6288 and C7552, it averages 15 to 30 seconds (CPU time) to find a multi-way

Table 2. *Comparison of the number of components, the total number of pins and cut nets*

Circuit	Area Limit	Pin Limit	TAPIR			FBB-MW			FBB-MW Improv.%		
			#comp.	#pins	#nets	#comp.	#pins	#nets	#comp.	#pins	#nets
c5315	1500	100	8	610	138	7	584	126	12.5	4.3	8.7
c7552	1500	100	5	432	55	5	441	61	0	-2.1	-10.9
c6288	1500	100	4	335	122	2	162	49	50	51.6	59.8
s5378	1500	100	8	663	254	6	567	221	25	14.48	12.9
s15850	1500	100	13	869	321	10	857	287	23	1.4	10.6
s9234	1500	100	7	493	197	6	441	162	14	10.5	17.6
s5378	3000	150	5	592	229	2	132	22	60	77.7	90.4
s15850	3000	150	7	669	250	5	604	238	28	9.7	4.8
s15850	3000	200	6	665	254	4	582	228	33	12.5	10.2
s15850	5000	200	4	562	214	3	417	157	25	25.8	26.6
s13207	5000	200	4	562	154	2	306	75	50	45.6	51.3
s35932	5000	200	8	1101	306	7	978	254	12.5	11.2	16.9
s38417	5000	200	9	760	280	6	847	244	33	-11.4	12.8
s35932	10000	250	5	957	270	3	553	95	40	42.2	64.8
s38417	10000	250	4	652	251	3	558	197	25	14.4	21.5
s38584	10000	250	5	960	311	3	570	134	40	40.6	56.9

Table 1. *Circuits in Partition93 Benchmark*

Circuit Name	# of Nodes	# of Nets	# of I/O
c5315	1778	1655	301
c7552	2247	2140	313
c6288	2856	2824	64
s5378	3225	3176	88
s9234	6098	6076	45
s13207	9445	9324	156
s15850	11071	10984	105
s35932	19882	19560	359
s38417	25589	25483	138
s38584	22451	20719	294

partition with area limit 1500 and pin limit 100. For large circuit s38417 which has 25589 nodes, the running time for multi-way partitioning under area limit 10000 and pin limit 250 is around 10 minutes CPU time. It takes about 20 minutes CPU time to partition s38417 under area limit 5000 and pin limit 200. The observation is that for the same circuit, it usually takes a longer running time when the area and pin limit becomes smaller. This is because with tighter constraints, fewer nodes can be packed in one component and therefore more components will be resulted and it takes a longer running time.

5. CONCLUSION

In this paper, we introduce algorithm FBB-MW, which is an extension of FBB[1], to multi-way partitioning with area and pin constraints. First, we presented an improvement to FBB by finding the most desirable min-cut in order to make better utilization of the min-cuts in the network.

Three network flow based algorithms for multi-way partitioning with area and pin limit are proposed. Algorithm 1 is a direct extension of FBB to multi-way partitioning. In algorithm 2, we give the net modeling of the I/O nodes so that the min-cut size is equal to the total number of pins for one component. By merging the first two algorithms, FBB-MW is a further improvement which produces better partitioning results than the FM based method in terms of the number of components, total number of pins and cut nets.

REFERENCES

[1] Honghua Yang and D.F. Wong, "Efficient Network Flow Based Min-Cut Balanced Partitioning", *Proc. ICCAD 1994*, pp50-55.

[2] Jianmin Li, John Lillis and Chung-Kuan Cheng, "Linear Decomposition Algorithm for VLSI Design Applications", *ICCAD'95*, pp223-228.

[3] Pak K. Chan, Martin D.F Schlag and Jason Y. Zien, "Spectral-Based Multi-Way FPGA Partitioning", *FPGA'95*, pp133-139, Monterey, CA.

[4] Dennis J.H Huang and Andrew B. Kahng, "Multi-Way System Partitioning into a Single Type or Multiple Types of FPGAs", *FPGA'95*, pp140-145, Monterey, CA.

[5] N.C. Chou, L.T. Liu, C.K. Cheung, W.J. Dai and R. Lindelof, "Circuit Partitioning for huge logic emulation systems", *31th ACM/IEEE Design Automation Conference*, pp244-249, CA, June 1994.

[6] R. Kuznar, F. Brglez and K. Kozminski, "Cost Minimization of Partitions into Multiple Devices", *30th ACM/IEEE Design Automation Conference*, 1993, pp315-320.

[7] N. S. Woo and J. Kim, "An Efficient Method of Partitioning Circuits for Multiple-FPGA Implementation", *30th ACM/IEEE Design Automation Conference*, pp202-207, Texas, June 1993.

[8] Minshine Shih, Ernest S. Kuh and Ren-Song Tsay, "Performance-Driven System Partitioning on Multi-Chip Modules", *29th ACM/IEEE Design Automation Conference*, 1992, pp53-56.

[9] J.R. Ford and D.R. Fulkerson, *Flows in Networks*, Princeton University Press, 1962.

[10] L. James Hwang and Abbas El Gamal, "Min-Cut Replication in Partitioned Networks", *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol 14, No. 1, pp96-106, Jan. 1995.

[11] Laura A. Sanchis, "Multiple-Way Network Partitioning", *IEEE Trans. on Computers*, VOL. 38, No.1, Jan. 1989.