Efficient Network Flow Based Min-Cut Balanced Partitioning

Hannah Honghua Yang and D. F. Wong

***Abstract*— We consider the problem of bipartitioning a circuit into two balanced components that minimizes the number of crossing nets. Previously, Kernighan and Lin type (K&L) heuristics, simulated annealing approach, and analytical methods were given to solve the problem. However, network flow (max-flow min-cut) techniques were overlooked as viable heuristics to min-cut balanced bipartition due to their high complexity. In this paper we propose a balanced bipartition heuristic based on repeated max-flow min-cut techniques, and give an efficient implementation that has the same asymptotic time complexity as that of one max-flow computation. We implemented our heuristic algorithm in a package called FBB. The experimental results demonstrate that FBB outperforms K&L heuristics and analytical methods in terms of the number of crossing nets, and our efficient implementation makes it possible to partition large circuit netlists with reasonable runtime. For example, the average elapsed time for bipartitioning a circuit S35932 of almost 20 K gates is less than 20 min on a SPARC10 with 32 MB memory.**

I. Introduction

CIRCUIT PARTITIONING is a fundamental problem in many areas of VLSI layout and design, such as floorplanning, placement and multiple-chip/multiple-FPGA partitioning. *Min-cut balanced bipartition* is the problem of partitioning a circuit into two disjoint components with equal weights such that the number of nets connecting the two components is minimized. The min-cut balanced bipartition problem was shown to be NP-complete [11]. Because of its importance, many heuristic algorithms have been devised for its solution. Among the well-known heuristics are the following [6]: Kernighan and Lin type (K&L) iterative improvement methods [20], [9], simulated annealing approaches [19], and analytical methods for the ratio-cut objective [25], see e.g., [15], [4], [23].

The well-known network max-flow min-cut theorem [8], [22], [7], [10], [16] is an important combinatorial optimization technique. It has many applications in VLSI design such as linear placement [3], min-cut replication [13], [14], and FPGA technology mapping [2], [27]. The network max-flow min-cut technique is in fact the most natural method for finding a min-cut in a graph. However, it was overlooked as a viable approach for circuit partitioning due to the following reasons: 1) The two components obtained by the network max-flow min-cut technique are not necessarily balanced, 2) Although a balanced cut can be achieved by repeatedly applying min-cut to the larger component, this method can possibly incur n max-flow computations, where n is the size of flow network, 3) The traditional network flow technique works on graphs, but hypergraphs are more accurate models for circuit netlists than graphs.

In this paper we explore solutions to the above problems faced by the traditional network flow technique. We first propose a method for exactly modeling a netlist (or equivalently, a hypergraph) by a flow network, and a balanced bipartition heuristic based on a repeated max-flow min-cut technique. We then give an efficient implementation of the repeated max-flow min-cut heuristic that has the same asymptotic time complexity as that of one max-flow computation.

We use a generalized notion of the balanced bipartition, the *r-balanced bipartition* (also used in [9]), which is a bipartition such that one component is of weight a fraction r of the total weight W. As a special case when $r = 1/2$, an r-balanced bipartition is a balanced bipartition. Since in practice there is little reason to strictly enforce the r-balanced criterion, we introduce a *deviation factor* ϵ to allow the component weight to deviate from $(1-\epsilon)rW$ to $(1+\epsilon)rW$. We show in Theorem III.2 that both the runtime and the cut size produced by our algorithm are decreasing functions of ϵ. This kind of direct relationship was not shown in previous partitioning heuristics.

The rest of this paper is organized as follows. In Section II, we first present a method for exactly modeling a netlist by a flow network, and an optimal algorithm for finding a min-net-cut bipartition (not necessarily balanced) of a circuit with respect to a source and a sink. This algorithm serves as a basic procedure for our min-cut balanced bipartition heuristic. We then present our heuristic algorithm for finding a min-net-cut r-balanced bipartition based on the repeated network flow technique in Section III with an efficient implementation that has the same asymptotic time complexity as that of computing one max-flow in a flow network. We compare our balanced bipartition results with those of K&L heuristics and analytical methods in Section IV, and conclude the paper in Section V.

II. An Optimal Algorithm for Min-Net-Cut Bipartition

In this section we first give some definitions of a flow network in Section II-A. We then show how to model a circuit using a flow network in Section II-B. Based on the flow network constructed in Section II-B, we present in Section II-C an optimal algorithm for finding a bipartition (not necessarily

Manuscript February 21, 1995; revised September 22, 1995 and August 30, 1996. This work was supported in part by the Texas Advanced Research Program under Grant 003658459, by an Intel Foundation Graduate Fellowship, by a DAC Design Automation Scholarship, and by a Grant from AT&T Bell Laboratories. This paper was recommended by Associate Editor C.-K. Cheng.

H. H. Yang is with Intel Development Laboratories, Intel Corporation, Hillsboro, OR 97124 USA.

D. F. Wong is with the Department of Computer Sciences, University of Texas, Austin, TX 78712 USA.

Publisher Item Identifier S 0278-0070(96)09412-2.

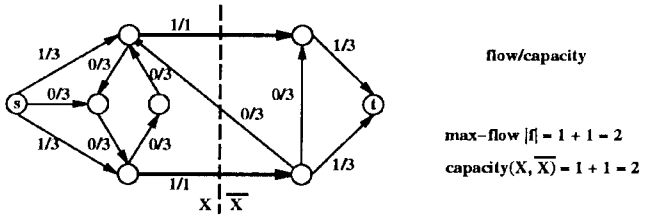

Fig. 1. A flow network G, and a max-flow f in G. The label x/y on an edge indicates that the flow and the capacity on the edge are x and y, respectively. The dark edges are the forward edges of the cut $(X, \bar{X})$.

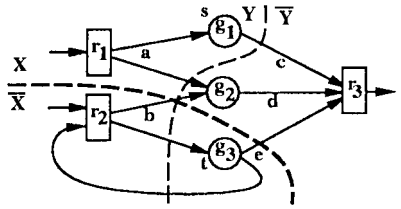

Fig. 2. A digraph N representing a sequential circuit and its net-cuts.

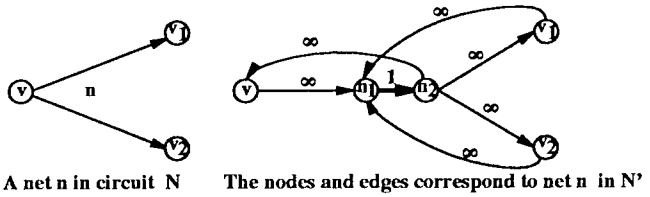

Fig. 3. Modeling a net in N in the flow network N'.

balanced) of a circuit that minimizes the number of crossing nets with respect to a source and a sink.

A. Preliminaries

A flow network $G = (V, E)$ is a directed graph in which each edge $e \in E$ has a *capacity* $c(e) \geq 0$. Two nodes s and t in V are specified: s is called the *source*, t is called the *sink*[1] (see Fig. 1). An *s-t flow* (or *flow* for short) in G is a real-valued function $f : E \rightarrow R$ such that (1) for all $e \in E$, $0 \leq f(e) \leq c(e)$, and (2) for all $u \in V \backslash \{s, t\}$, the sum of the incoming flow into u is equal to the sum of the outgoing flow from u. An edge e in E is *saturated* if $f(e) = c(e)$. The *value* $|f|$ of a flow f is defined as the sum of the flow outgoing from s, which is equal to the sum of the flow incoming to t. A *maximum-flow* (or *max-flow* for short) in G is a flow of maximum value from s to t.

An *s-t cut* (or *cut* for short) $(X, \bar{X})$ of a flow network $G = (V, E)$ is a bipartition of V into X and $\bar{X}$ such that $s \in X$ and $t \in \bar{X}$. An edge whose starting node is in X and ending node is in $\bar{X}$ is called a *forward edge*. An edge whose ending node is in X and starting node is in $\bar{X}$ is called a *backward edge*. The *capacity of the cut* $(X, \bar{X})$, denoted by $cap(X, \bar{X})$, is the sum of the capacities on the *forward edges only* from X to $\bar{X}$.[2] An *augmenting path* from u to v in G is a simple path from u to v in the undirected graph resulting from the network by ignoring edge directions, that can be used to push additional flow from u to v.

Theorem II.1— Max-flow min-cut theorem [8]: Given a max-flow f in G, let $X = \{v \in V : \exists$ an augmenting path from s to v in $G\}$, and let $\bar{X} = V \backslash X$. Then $(X, \bar{X})$ is a cut of minimum capacity (which is equal to $|f|$), and f saturates all forward edges from X to $\bar{X}$.

Fig. 1 shows an example of a max-flow in G and the corresponding cut of minimum capacity.

B. Modeling a Net in a Flow Network

We represent a *sequential circuit* netlist as a digraph (which may contain directed cycles) $N = (V, E)$ where V is a set of nodes representing combinational gates and registers, and E is a set of edges representing interconnections between gates and registers. Each node v in V has an associated weight $w(v) \in R^+$. The total weight of a subset $U \subseteq V$ is denoted by $w(U)$. Let $W = w(V)$ denote the total weight of the circuit N.

A *net* $n = (v; v_1, \ldots, v_l)$ is a set of outgoing edges from node v in N. For example in Fig. 2, net a consists of two edges (r_1, g_1) and (r_1, g_2). Given two nodes s and t in N, an *s-t cut* (or *cut* for short) $(X, \bar{X})$ of N is a bipartition of the nodes in V such that $s \in X$ and $t \in \bar{X}$. The *net-cut* *net*$(X, \bar{X})$ of the cut is the set of nets in N that are incident to nodes in both X and $\bar{X}$. In Fig. 2, if we choose $s = g_1$ and $t = g_3$, then the net-cut *net*$(Y, \bar{Y})$ corresponding to the cut $(Y, \bar{Y})$ consists of nets c, a, b, e, where nets c, a, b connect nodes from Y to $\bar{Y}$, and net e connects nodes from $\bar{Y}$ to Y. A cut $(X, \bar{X})$ is a *min-net-cut* if $|net(X, \bar{X})|$ (i.e., the number of nets in *net*$(X, \bar{X})$) is minimum among all *s-t* cuts of N. In Fig. 2, *net*$(X, \bar{X}) = \{b, e\}$, *net*$(Y, \bar{Y}) = \{c, a, b, e\}$, and $(X, \bar{X})$ is a min-net-cut.

In order to find a min-net-cut in $N = (V, E)$, we reduce it to the problem of finding a cut of minimum capacity, and then solve the latter problem by the max-flow min-cut theorem. If the cut edges all have unit capacity, then the problem is equivalent to finding a cut with the minimum number of forward edges from X to $\bar{X}$.

We construct a flow network $N' = (V', E')$ from $N = (V, E)$ as follows (see Figs. 3 and 4):

- V' contains all nodes in V.
- for each net $n = (v; v_1, \ldots, v_l)$ in N, add two nodes n_1 and n_2 in V' and a *bridging edge* (n_1, n_2) in E';
- for each node $u \in \{v, v_1, \ldots, v_l\}$ incident on net n, add two edges (u, n_1) and (n_2, u) in E';
- let s be the source of N' and t the sink of N';
- assign unit capacity to all bridging edges and infinite capacity to all other edges in E';
- for a node $v \in V'$ corresponding to a node in V, $w(v)$ is the weight of v in N. For a node $u \in V'$ split from a net, $w(u) = 0$.

Note that all nodes incident on net n are connected to n_1 and are connected from n_2 in N'. Hence the flow network construction is symmetric with respect to all nodes incident on a net. We show in Lemma II.1 that the size of N' is only

[1] The choice of s and t is completely arbitrary. There is no requirement that s has no incoming edges, or that t has no outgoing edges. The edges entering s or leaving t are actually redundant and have no effect on our problem, but we allow them in G since the choice of s and t may vary.

[2] Note that the capacities on the backward edges from $\bar{X}$ to X are not included.

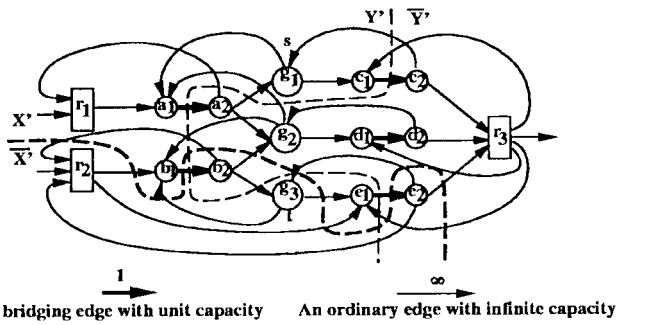

Fig. 4. A flow network N' constructed from the circuit N in Fig. 2.

a constant factor larger than the size of N, for a connected graph N.

Lemma II.1: Let $N' = (V', E')$ be the flow network constructed from a digraph $N = (V, E)$ using the above method. Then $|V'| \leq 3|V|$ and $|E'| \leq 2|E| + 3|V|$.

Proof: The number of nets in N, denoted by *nets*(N), is less than or equal to $|V|$, since the set of outgoing edges from each node forms a net.[3] V' contains all nodes in V, and two nodes for each net in N. Hence $|V'| = |V| + 2$*nets*$(N) \leq 3|V|$. For each net $n = (v; v_1, \ldots, v_m)$ in N consisting of m edges in E, there are $m + 1$ edges incoming to n_1, $m + 1$ edges outgoing from n_2, and a bridging edge (n_1, n_2) in N'. Hence $|E'| = 2(|E| + $ *nets*$(N)) + $ *nets*$(N) = 2|E| + 3$*nets*$(N) \leq 2|E| + 3|V|$. □

The above flow network construction for modeling net-cuts also works when the circuit N is represented by a hypergraph. We note that another optimal approach for finding a min-net-cut of a hypergraph was given in [16] by modeling a net (or a hyperedge) as a star node, and then transforming a node-capacitated flow network into an edge-capacitated network [22] by splitting *every* node. Our method is different from that of [16] in that we split the nodes corresponding to the nets *only*, and hence use fewer nodes and edges in the resulting flow network. For a huge input circuit, our method translates into less memory usage and faster runtime.

Another related result given in [18] shows that modeling hypergraphs by graphs (with positive weights) with the same min-cut properties is not possible. However, our method models a hypergraph for network flow based partition algorithms only. The differences between the model used in [18] and our model are the following: 1) The weight of a cut in [18] is computed by the sum of all cut edges with fixed weights, while the weight of a cut in our model is computed by the sum of the capacities of the forward edges only. 2) Reference [18] tries to model hypergraphs for a wide range of existing partition algorithms developed for ordinary graphs only, while our method just models a hypergraph for network flow based partition algorithms. Hence our method is able to exactly model a hypergraph for our flow based partitioning algorithm without contradicting the result in [18].

[3] In this proof we assume that each node has one output net, and each net feeds at most one input on another node. This assumption is reasonable for synthesized circuits where a combinational gate has one output net, and a flip-flop has two which is still constant bound. It is very rare for a net to feed more than one input on a node and it is not considered a good design style.

It is easy to see that N' is a strongly connected digraph. The strong connectivity of N' is the key to reducing the bi-directional min-net-cut problem to the minimum capacity cut problem that counts the capacity of the forward edges only. We show in Theorem II.2 and Corollary II.2.1 that the problem of finding a min-net-cut in N can be reduced to the problem of finding a cut with minimum capacity in N'.

Theorem II.2: N has a cut of net-cut size at most C if and only if N' has a cut of capacity at most C.

Proof: 1) Assume $(X, \bar{X})$ is a cut in N with net-cut size $|$*net*$(X, \bar{X})| \leq C$. We define a cut $(X', \bar{X}')$ in N' as follows. Let X' contain all nodes in X, the two split nodes for each net residing entirely in X (i.e., all nodes incident on the net are in X), and the nodes n_1 where n is a net in the net-cut *net*$(X, \bar{X})$. Let $\bar{X}'$ contain the rest of the nodes in N'. Then for each net $n \in$ *net*$(X, \bar{X})$, we have $n_1 \in X'$ and $n_2 \in \bar{X}'$, and therefore the bridging edge (n_1, n_2) is a forward edge from X' to $\bar{X}'$ in N'. Further, the bridging edge (b_1, b_2) of a net $b \notin$ *net*$(X, \bar{X})$ will not be a forward edge in N' since b_1 and b_2 either both reside in X' or both reside in $\bar{X}'$. Suppose there exists a forward edge (u, v) where $u \in X'$ and $v \in \bar{X}'$ in N' that is not a bridging edge. Then either $u = b_2$ for some net b in N or $v = c_1$ for some net c in N. Without loss of generality, assume $u = b_2$ for some net b in N and v corresponds to a node in $\bar{X}$. Since $u = b_2$ is in X', by the definition of $(X', \bar{X}')$ net b must have resided entirely in X. But $v \in \bar{X}$ is a node incident on net b, a contradiction. Hence the bridging edges corresponding to the nets in *net*$(X, \bar{X})$ are exactly the forward edges from X' to $\bar{X}'$. Since the bridging edges have unit capacity, the capacity of cut $(X', \bar{X}') = |$*net*$(X, \bar{X})| \leq C$.

2) Conversely, assume $(X', \bar{X}')$ is a cut of capacity $\leq C$ in N'. Then all forward edges from X' to $\bar{X}'$ must be bridging edges since the capacities of the nonbridging edges are infinite, and the number of forward edges from X' to $\bar{X}'$ is C since all bridging edges have unit capacity. If we remove all nets in N corresponding to the forward (bridging) edges from X' to $\bar{X}'$, then N will be divided into several disjoint components and s and t are in different components. Let X be the component containing s, and let $\bar{X}$ be the union of the other components. Then the net-cut size of $(X, \bar{X})$ is at most the number of nets removed. The number of nets removed is the number of forward (bridging) edges from X' to $\bar{X}'$, which is at most C. Hence $|$*net*$(X, \bar{X})| \leq C$. □

Corollary II.2.1: Let $(X', \bar{X}')$ be a cut of minimum capacity C in N', and let $(X, \bar{X})$ be the cut in N as constructed in Theorem II.2 (2). Then $(X, \bar{X})$ is a min-net-cut in N, and $|$*net*$(X, \bar{X})| = C$.

Proof: Suppose there is a cut in N of net-cut size $C' < C$. By Theorem II.2-1), there is a cut in N' of capacity at most C'. This contradicts the fact that $(X', \bar{X}')$ is a cut of minimum capacity C in N'. □

C. Optimal Network Flow Based Min-Net-Cut Bipartition

As a result of the correspondence between hypergraphs and flow networks, one can find a min-net-cut in a circuit.

Algorithm 1: Finding A Min-Net-Cut

0) Construct the flow network $N' = (V', E')$ for N as described in Section II-B;

1) Find a max-flow in N' from s to t;
2) Find a cut $(X', \bar{X}')$ of minimum capacity in N' as described in the max-flow min-cut theorem;
3) Find a min-net-cut $(X, \bar{X})$ in N as described in Theorem II.2-2).

Theorem II.3: Algorithm 1 finds a min-net-cut in a circuit $N = (V, E)$, and terminates in $O(|V||E|)$ time where N is a connected circuit.

Proof: The correctness of Algorithm 1 has been established in Theorem II.2 and Corollary II.2.1.

Steps 0, 2, and 3 take linear time in the size of N. We use the simple augmenting path algorithm [8] to implement step 1. Finding an augmenting path in N' takes $O(|E'|)$ time. The number of augmenting paths in N' is equal to the number of bridging edges in a min-net-cut, which is at most $|V|$ (usually much less). Hence Algorithm 1 takes $O(|V||E'|) = O(|V|(|E| + |V|)) = O(|V||E|)$ time by Lemma II.1. □

There are other asymptotically faster (worse-case) algorithms for finding a max-flow than the simple augmenting path algorithm based on the Ford and Fulkerson method. The fastest preflow method takes $O(|E||V| \log(|V|^2/|E|))$ time [12] with a large constant factor. The Ford and Fulkerson method takes $O(|E| * \text{max-flow-value})$ time. The latter method is efficient in our application, since the max-flow-value in the special flow network we construct is at most $|V|$ (usually much smaller than $|V|$).

The min-cut bipartition may yield unbalanced components. The max-flow computation defines a set of min-cuts with the same cut size, but with varying weights in the two partitions. It is natural to ask the question of whether one can find a min-cut that is the *most r-balanced* among all the min-cuts defined by a max-flow, i.e., among all possible min-cuts $(X, \bar{X})$ defined by a max-flow find the min-cut such that $|w(X) - rw(V)|$ is as close to 0 as possible. One can show that the decision version of the problem is NP-complete by reducing the weighted subset sum problem [11] to it [26].

III. Min-Cut Balanced Bipartition

It is not difficult to see that repeatedly applying the max-flow min-cut technique to cut the larger of the two partitions will eventually produce a balanced bipartition with a natural small net-cut. However, this approach was overlooked as a viable heuristic approach to circuit partitioning due to its high complexity (possibly $|V|$ max-flow computations). In this section we first describe a repeated max-flow min-cut heuristic algorithm, Flow-Balanced-Bipartition (FBB), for finding an r-balanced bipartition that minimizes the number of crossing nets. We then give an efficient implementation of FBB that has the same asymptotic time complexity as one max-flow computation. For ease of presentation, we will describe our algorithm in terms of the original circuit and net-cuts, instead of the flow network constructed from the circuit (as shown in Section II-B) and forward bridging edges, when there is no confusion. An illustration of Algorithm 2 is shown in Fig. 5.

A. Balanced Bipartition Heuristic

Given a circuit $N = (V, E)$, FBB randomly picks a pair of nodes s and t in N, and then tries to find an r-balanced bipartition that separates s and t, and that minimizes the number of crossing nets. Let W be the total weight of the circuit N. Since in practice there is little reason to strictly enforce the r-balanced criterion, we allow the component weights to deviate from $(1 - \epsilon)rW$ to $(1 + \epsilon)rW$. Given a subcircuit X of N, let $w(X)$ denote the total weight of nodes in X.

Algorithm 2: Flow-Balanced-Bipartition (FBB)

0) Randomly pick a pair of nodes s and t in N;
1) Find a min-net-cut C in N; Let X be the subcircuit reachable from s through augmenting paths in the flow network, and $\bar{X}$ the rest;
2) If $(1 - \epsilon)rW \leq w(X) \leq (1 + \epsilon)rW$ then stop and return C as the answer.
3) If $w(X) < (1 - \epsilon)rW$
   then 3.1 collapse all nodes in X to s;
   3.2 collapse to s a node $v \in \bar{X}$ adjacent to C;
   3.3 goto 1;
4) If $w(X) > (1 + \epsilon)rW$ then
   then 4.1 collapse all nodes in $\bar{X}$ to t;
   4.2 collapse to t a node $v \in X$ adjacent to C;
   4.3 goto 1.

Step 1 can be implemented by Algorithm 1. In step 3.2, we need to collapse a node $v \in \bar{X}$ incident on a cut net to s since otherwise the same set of nets in C will again be chosen as the min-net-cut in the next iteration in step 1. The reasons why we adopt this node collapsing method instead of a more gradual method (i.e., increasing the unit capacities of the bridging edges by a fixed amount as in [17] are (1) the capacity of the cut would no longer reflect the real net-cut size, and (2) the runtime would not be bounded by one flow computation. By collapsing v to s, FBB is able to explore a different net-cut with a *larger* X in the next iteration. Note that the size of the min-net-cut found in the next iteration will be the same as or larger than the size of the min-net-cut in the current iteration. A similar argument holds for step 4.2.

We now describe our strategy for picking a node in steps 3.2 and 4.2. To find an r-balanced bipartition that minimizes the net-cut size, our heuristic is to always focus on finding a min-net-cut at each iteration. But when the remaining circuit is very large, the current min-net-cut has less influence on what the final balanced min-net-cut would be. Therefore, we randomly pick a node in steps 3.2 and 4.2 in order to speed up the algorithm. When the remaining circuit becomes small enough, we need to be more careful about which node we pick, and we can afford to try out more than one node. We give a threshold value R for the number of nodes in the un-collapsed subcircuit. If the number of remaining nodes is larger than R, then we randomly pick one node from the nodes incident on the cut nets in C. Otherwise, we try all nodes incident on the cut nets in C and pick the node whose collapsing induces a min-net-cut with the smallest size. We can also let the probability of choosing a node be inversely proportional to the number of nodes in the remaining (un-collapsed) circuit.

B. Efficient Implementation of Algorithm 2

A drawback of the repeated max-flow heuristic is that it has a relatively high time complexity. Iteratively applying

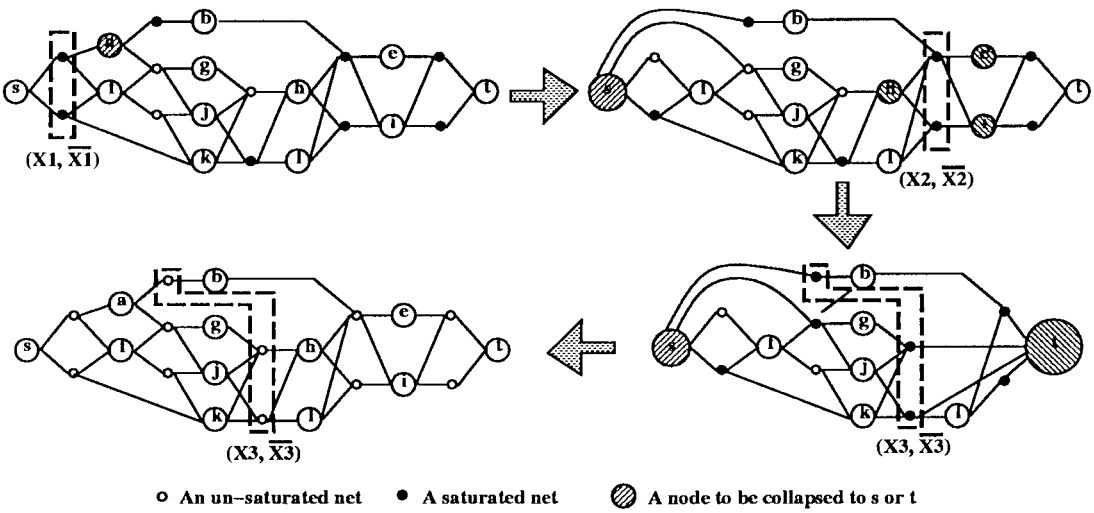

Fig. 5. An illustration of Algorithm 2 for $r = 1/2$, $\epsilon = 0.15$ and unit weight for each node. Hence a component can have weight between 5 and 7. If $\epsilon = 0.45$, then a component can have weight between 3 and 10, and Algorithm 2 would terminated after finding cut $(X_2, \bar{X}_2)$. A small solid node indicates that the bridging edge corresponding to the net is saturated with flow.

Algorithm 1 in step 1 of Algorithm 2 to compute a max-flow and a min-net-cut from the zero flow can be very time-consuming. We show an efficient way to deal with this problem. In fact, it is not necessary to do the max-flow computation from the zero flow in every iteration. Instead, we can retain the flow value in the flow network, and only find additional flow to saturate the bridging edges of the min-net-cut from iteration to iteration.

In Procedure 1, we describe the incremental max-flow computation in step 1 of Algorithm 2. Initially, the flow network retains the flow function computed in the previous iteration.

Procedure 1: Incremental Flow Computation

0) While $\exists$ an additional augmenting path from s to t
1) increase flow value along the augmenting path;
   /* There is no more augmenting path from s to t.*/
2) Mark all nodes u such that
   $\exists$ an augmenting path from s to u;
3) Let C' be the set of bridging edges whose starting nodes are marked and ending nodes are not marked;
4) Return the nets corresponding to the bridging edges in C' as the min-net-cut C, and the marked nodes as X

Since the max-flow computation using the augmenting path method is insensitive to the initial flow values in the flow network and the order in which the augmenting paths are found, the above procedure correctly finds a max-flow with the same flow value as a max-flow computed in the collapsed flow network from scratch (i.e., the zero flow).

We show in Theorem III.1 that if we fix the threshold R used in the node-picking strategy described in the previous subsection as a constant, then the total complexity of FBB is $O(|V||E|)$, which is the same as the complexity of one max-flow computation.

Theorem III.1: If Procedure 1 is used to implement step 1 of Algorithm 2, then Algorithm 2 has time complexity $O(|V||E|)$ for a connected circuit $N = (V, E)$.

Proof: Since each augmenting path computation takes $O(|E|)$ time, we prove that the total time complexity of step 1 of Algorithm 2 is $O(|V||E|)$ by showing that there are at most $2|V|$ augmenting path computations in the following iterations.

The total flow value $|f|$ in the flow network N' constructed from N at the end of Algorithm 2 is the number of forward bridging edges in the final min-net-cut. Hence $|f|$ is at most $|V|$. Since bridging edges have unit capacity, there are $|f| \leq |V|$ s-t augmenting paths found at the end of Algorithm 2, We now consider an augmenting path computation in step 0 of Procedure 1. Either an augmenting path is found, in which case the number of augmenting paths increases by 1, or at least one node will be collapsed to s or t in steps 3.1 and 3.2 or 4.1 and 4.2 of Algorithm 2. Hence the number of augmenting path computations in the following iterations is at most $2|V|$.

Note that step 3 of Procedure 1 can be accomplished during the searching for an augmenting path in step 2 of Procedure 1, and steps 4 and 5 of Procedure 1 takes $O(|V|)$ time in the worst case. □

In practice, as shown in the experimental results, Algorithm 2 terminates much faster than the $O(|V||E|)$ worst case time complexity. Because of the construction of the flow network in Section II-B where the bridging edges have unit capacity, the number of augmenting paths found in Algorithm 2 is the same as the size of the net-cut found in N, which is much less than $|V|$.

Theorem III.2: The number of iterations and the final net-cut size of Algorithm 2 are nonincreasing functions of ϵ.

Proof: Fewer iterations are needed in Algorithm 2 when ϵ is larger, since the condition in step 2 of Algorithm 2 is satisfied in fewer iterations.

If an augmenting path from s to t is found in step 1 of Procedure 1, then the flow value is increased by at least 1 and hence the size of the min-net-cut is increased by at least 1. If an augmenting path from s to t is not found in step 1 of Procedure 1, then the size of the min-net-cut is equal to the flow value of the previous iteration, which is equal to the previous min-net-cut size. Hence the net-cut size found in each iteration is nondecreasing. □

TABLE I
COMPARISON OF BIPARTITION RESULTS OF SN, PFM3, AND FBB (WITH $r = 1/2$ AND $\epsilon = 0.1$)

circuit				ave. net-cut size			ave. FBB	FBB improvem. %	
name	gates & latches	nets	ave. net degree	SN	PFM3	FBB	bipart. ratio	over SN	over PFM3
C1355	514	523	3.0	38.9	29.1	26.0	1:1.08	33.2	10.7
C2670	1161	1254	2.6	51.9	46.0	37.1	1:1.15	28.5	19.3
C3540	1667	1695	2.7	90.3	71.0	79.8	1:1.11	11.6	-12.4
C7552	3466	3565	2.7	44.3	81.8	42.9	1:1.08	3.2	47.6
S838	478	511	2.6	27.1	21.0	14.7	1:1.04	45.8	30.0
Average							1:1.10	24.5	19.0

TABLE II
COMPARISON OF BIPARTITION RESULTS OF EIG1, PARABOLI, AND FBB (WITH $r = 1/2$ AND $\epsilon = 0.1$). ALL RESULTS ALLOW UP TO 10% DEVIATION FROM BISECTION

circuit				best net-cut size			FBB improvem. % over		FBB
name	gates & latches	nets	ave. net degree	EIG1	PARABOLI	FBB	EIG1	PARABOLI	elapsed time (sec.)
S1423	731	743	2.7	23	16	13	43.5	18.8	1.7
S9234	5808	5805	2.4	227	74	70	69.2	5.4	55.7
S13207	8696	8606	2.4	241	91	74	69.3	18.9	100.0
S15850	10310	10310	2.4	215	91	67	68.8	26.4	96.5
S35932	18081	17796	2.7	105	62	49	53.3	21.0	2808
S38584	20859	20593	2.7	76	55	47	38.2	14.5	1130
S38417	24033	23955	2.4	121	49	58	52.1	-18.4	2736
Average							58.5	11.3	

Theorem III.2 guarantees that with a larger ϵ deviation factor we can improve the efficiency of Algorithm 2 and obtain a better partitioning solution. This property is not true for other partitioning approaches such as the K&L heuristics. Another interesting corollary of Theorem III.2 is that the longer the execution time of Algorithm 2, the worse the net-cut size in the final solution. This property of Algorithm 2 can be utilized to further improve the efficiency of Algorithm 2.

IV. EXPERIMENTAL RESULTS

We implemented Algorithm 2 in a package called FBB using the C language, and integrated FBB in SIS/MISII [1]. Currently FBB runs on circuit formats accepted by SIS/MISII. We tested FBB on a set of large ISCAS and MCNC benchmark circuits using a SPARC 10 workstation with a 36 MHz SS10 and 32 MB memory (the C code was compiled with gcc without the optimizer). For each circuit tested, the number of gates and latches, the number of nets[4] and the average net degree (i.e., the average number of nodes connected to a net) are given in Tables I and II. Note that the actual number of nodes in a circuit includes PI nodes, and is therefore more than the the number of gates and latches.

Table I compares the average bipartition results of FBB with the results reported by Dasdan and Aykanat in [5]. The program SN is based on the K&L heuristic algorithm in Sanchis [24], which is a generalization of the Krishnamurthy [21] algorithm. The program PFM3 is based on a K&L heuristic with free moves as described in [5]. SN was run 20 times and PFM3 was run 10 times on each circuit starting from different randomly generated initial partitions, while FBB was run 10 times on each circuit from different randomly generated s and t as the source and the sink respectively. Table I shows that with only one exception, FBB outperforms both SN and PFM3 on the 5 circuits. On average, FBB finds a bipartition with 24.5 and 19.0% fewer crossing nets than SN and PFM3, respectively. This is not too surprising since max-flow min-cut techniques tends to find a natural small cut. The average actual ratios of the two partitions obtained by FBB are also shown in Table I. Since we set $\epsilon = 0.1$, the actual ratios of the two partitions are roughly the same (1:1.10 on average).

We did not compare the runtime of SN, PFM3, and FBB since they were run on different workstations. SN and PFM3 were run on a SUN SPARC ELC, and FBB were run on a SUN SPARC 10. For example, for C3540, the average elapsed time (not CPU time) in seconds of SN, PFM3, and FBB for each run are 90.3, 71.0, and 13.6, respectively; and for C7552, the average elapsed time in seconds of SN, PFM3, and FBB for each run are 44.3, 81.8, and 18.8, respectively.

Table II compares the best bipartition net-cut size of EIG1 (Hagen and Kahng [15]), PARABOLI (Riess, Doll, and Frank [23]), and FBB. EIG1 and PARABOLI are two programs based on analytical methods and their results were obtained from [23]. The results produced by PARABOLI were the best previously known results reported on the benchmark circuits. The results for FBB were the best of 10 runs. The

[4] Although we do not count a PI node in the second column in Tables I and II, we do consider the set of nodes receiving a fanin from the same PI node as being in a net.

elapsed time of FBB for the run that generates the best result was also recorded. All results in Table II allow up to 10% deviation from bisection. On average, FBB outperforms EIG1 and PARABOLI by 58.1% and 11.3%, respectively. For circuit S38417, FBB produces a larger net-cut than PARABOLI does. We consider the following possible explanations: 1) If FBB is run more than 10 times, the best net-cut result is likely to be better. 2) In a huge circuit like S38417, the solution is sensitive to the selection of the initial s, t pair of nodes. Applying circuit clustering techniques based on the connectivity information before partitioning may improve the partitioning result of FBB.

Note that different programs using the same MCNC benchmark circuits reported different properties such as the number of cells and the number of nets for these circuits. This is because when a netlist format is translated to a hypergraph, some unnecessary details such as inverters are omitted. However, the underlying netlist structures are the same.

In the experiment, we have consistently observed that for the runs with longer-than-average runtime, FBB always generates exceptionally poor solutions. This can be explained by Theorem III.2, since the net-cut size is nondecreasing with more iterations. This property of FBB is in contrast to both the K&L heuristics and the simulated annealing heuristics, where longer runtime means better solutions. This property of FBB provides another way of improving the efficiency of FBB. We can pick a reasonable upperbound for the runtime of FBB (for example, based on a few runs of FBB), stop FBB when the runtime exceeds the upperbound, and restart FBB using a new pair of nodes s and t. By doing so we are not likely to lose any good solutions, but we will further improve the efficiency of FBB.

V. Conclusion and Discussions

We have described a method for exactly modeling a netlist by a flow network, presented a balanced bipartition heuristic based on the repeated max-flow min-cut techniques, and given an efficient implementation of a good theoretical method. We implemented our algorithm in a package called FBB. The experimental results demonstrate that the repeated max-flow min-cut heuristic outperforms K&L heuristics and analytical methods in terms of the number of crossing nets, and the efficient implementation enables our heuristic algorithm to partition large benchmark circuits with reasonable runtime.

FBB has predictable behavior in terms of the sizes of the two partitions, and the direct relationship between efficiency, solution quality of FBB, and relaxing the r-balanced criterion by using a larger ϵ. Such a direct relationship was not shown in previous heuristics for circuit partition. We also believe that the choice of the pair of nodes s and t as the initial configuration of FBB has less influence on the solution than an initial bipartition would have. Hence the solution quality of FBB is less sensitive to the initial choice of s and t.

Our algorithm can be easily extended to handle that case where the nets in a circuit have different weights. We can simply assign the weight of a net to its corresponding bridging edge in the flow network, and FBB will find a net-cut with its weight minimized. K-way min-cut partitioning for $K > 2$ can be accomplished by recursively applying FBB, or by setting $r = 1/K$ and then using FBB to find one partition at a time. We are currently investigating more natural methods based on flow networks for the K-way min-cut partitioning problem.

Pre-partitioning circuit clustering according to the connectivity or the timing information of the circuit can be easily incorporated into FBB by treating a cluster as a node. A possible extension to FBB would be to combine FBB with the K&L heuristics and the simulated annealing heuristics. We can use FBB to find a natural small net-cut as a good initial partition, and then apply the K&L heuristics or the simulated annealing heuristics with low temperature to fine-tune the solution.

Acknowledgment

The authors would like to thank the referees for their suggestions.

References

[1] R. K. Brayton, R. Rudell, and A. L. Sangiovanni-Vincentelli, "MIS: A multiple-level logic optimization," *IEEE Trans. Computer-Aided Design*, pp. 1061–1081, Nov. 1987.
[2] J. Cong and Y. Ding, "An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 48–53.
[3] C.-K. Cheng, "Linear placement algorithms and applications to VLSI design," *Networks*, vol. 17, pp. 439–464, 1987.
[4] J. Cong, L. Hagen, and A. Kahng, "Net partitions yield better module partitions," in *Proc. 29th ACM/IEEE Design Automation Conf.*, 1992, pp. 47–52.
[5] A. Dasdan and C. Aykanat, "Improved multiple-way circuit partitioning algorithms," in *Int. ACM/SIGDA Wkshp. Field Programmable Gate Arrays*, Feb. 1994.
[6] W. E. Donath, "Logic partitioning," in *Physical Design Automation of VLSI Systems*, Preas and Lorenzetti, Eds. Menlo Park, CA: Benjamin/Cummins, 1988, pp. 65–86.
[7] S. Even, *Graph Algorithms*. Potomac, MD: Computer Science Press, 1979.
[8] J. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton Univ. Press, 1962.
[9] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175–181.
[10] R. Gomory and T. C. Hu, "Multi-terminal network flows," *J. SIAM*, vol. 9, pp. 551–570, 1961.
[11] M. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Fransisco, CA: W. H. Freeman, 1979.
[12] A. W. Goldberg and R. E. Tarjan, "A new approach to the maximum flow problem," *J. SIAM*, vol. 35, pp. 921–940, 1988.
[13] J. Hwang and A. El Gamal, "Optimal replication for min-cut partitioning," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 432–435.
[14] J. Hwang and A. El Gamal, "Min-Cut replication in partitioned networks," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 96–106, Jan. 1995.
[15] L. Hagen and A. B. Kahng, "Fast spectral methods for ratio cut partitioning and clustering," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1991, pp. 10–13.
[16] T. C. Hu and K. Moerder, "Multiterminal flows in a hypergraph," in *VLSI Circuit Layout: Theory and Design*, Hu and Kuh, Eds. New York: IEEE Press, 1985, pp. 87–93.
[17] S. Iman, M. Pedram, C. Fabian, and J. Cong, "Finding uni-directional cuts based on physical partitioning and logic restructuring," in *4th ACM/SIGDA Physical Design Wkshp.*, Apr. 1993.
[18] E. Ihler, D. Wagner, and F. Wager, "Modeling hypergraphs by graphs with the same min-cut properties," in *Info. Proc. Lett.*, vol. 45, pp. 171–175, 1993.
[19] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Sci.*, pp. 671–680, May 1983.

[20] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning of electrical circuits," *Bell Syst. Tech. J.*, pp. 291–307, Feb. 1970.
[21] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. on Computers*, pp. 438–446, May 1984.
[22] E. Lawler, *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart, and Winston, 1976.
[23] B. M. Riess, K. Doll, and M. J. Frank, "Partitioning very large circuits using analytical placement techniques," in *Proc. 31th ACM/IEEE Design Automation Conf.*, 1994, pp. 646–651.
[24] L. A. Sanchis, "Multiway network partitioning," *IEEE Trans. Comput.*, pp. 62–81, Jan. 1989.
[25] Y. C. Wei and C. K. Cheng, "Toward efficient hierarchical designs by ratio cut partitioning," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1989, pp. 298–301.
[26] H. Yang, "Algorithms for VLSI circuit partitioning," Ph.D. dissertation, The University of Texas at Austin, Aug. 1995.
[27] H. Yang and D. F. Wong, "Edge-Map: Optimal performance driven technology mapping for iterative LUT based FPGA designs," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 150–155.

Hannah Honghua Yang received the B.S. degree in computer science from Peking University, Beijing, China, in 1988, and the M.S. and Ph.D. degrees in computer science from the University of Texas at Austin in 1991 and 1995 respectively.

Since 1995, she has been with Intel Corporation, Hillsboro, OR. Her research interests include computer-aided design of VLSI circuits and combinatorial and parallel algorithms.

D. F. Wong received the B.Sc. degree from the University of Toronto and the M.S. degree from the University of Illinois at Urbana-Champaign, both in mathematics. He received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1987.

He is currently an Associate Professor of computer sciences with the University of Texas at Austin. His main research interest is CAD of VLSI and has published more than 100 technical papers in this area. He is a coauthor of *Simulated Annealing for VLSI Design* (Kluwer Academic: 1988).

Dr. Wong received the best paper awards for his work on floorplan design and FPGA routing, at DAC'86 and ICCD'95, respectively. He is currently serving on the editorial board of the IEEE TRANSACTIONS ON COMPUTERS.