Efficient Network Flow Based Min-Cut Balanced Partitioning *

Honghua Yang and D. F. Wong

Department of Computer Sciences, University of Texas at Austin, Austin, Texas 78712

Abstract

We consider the problem of bipartitioning a circuit into two balanced components that minimizes the number of crossing nets. Previously, the Kernighan and Lin type (K&L) heuristics, the simulated annealing approach, and the spectral method were given to solve the problem. However, network flow techniques were overlooked as a viable approach to min-cut balanced bipartition due to its 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 the K&L heuristics and the spectral method in terms of the number of crossing nets, and the efficient implementation makes it possible to partition large circuit instances with reasonable runtime. For example, the average elapsed time for bipartitioning a circuit S35932 of almost 20K gates is less than 20 minutes.

1 Introduction

Circuit partition is a fundamental problem in many areas of VLSI layout and designs, such as floorplanning, placement and multiple-chip/multiple-FPGA partition. The *min-cut balanced bipartition* is the problem of partitioning a netlist of 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 was shown to be NP-complete [GJ79]. Because of its importance, many heuristic algorithms have been devised to solve the problem. Among the well-known heuristics are the following ([Don88]): the Kernighan and Lin [KL70, FM82] type (K&L) group migration method, the simulated annealing approach [KGV83], and the spectral method for ratio-cut objective, see e.g., [HK91, CHK92, RDF94].

Network flow (max-flow min-cut) techniques [FF62, Law76, Eve79, HM85] have been known to find a min-cut bipartition (not necessarily balanced) in polynomial time. Repeatedly applying the max-flow min-cut technique will eventually produce a balanced bipartition. This approach is based on an elegant max-flow min-cut theory. However, it was overlooked as a viable approach to min-cut balanced partition since its high complexity makes it impractical for large problem instances. In this paper we propose a balanced bipartition heuristic based on repeated max-flow min-cut techniques. 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, instead of possibly n repeated max-flow computations, where n is the number of nodes in the circuit. Our experimental results show that on average, our repeated max-flow min-cut heuristic finds a bipartition with 24.5% fewer crossing nets than an implementation of the K&L heuristics, and 58.5% fewer crossing nets than an implementation of the spectral method, and that our efficient implementation makes it possible to partition large circuit instances with reasonable runtime.

We use a generalized notion of the balanced bipartition, the *r-balanced bipartition* (also used in FM82), which is a bipartition such that one component is of weight a fraction r of the total weight W. 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 3.2 that both the runtime and the cut size produced by our algorithm are decreasing functions of ϵ. This kind of direct relation was not shown in previous partition heuristics.

The rest of this paper is organized as follows. In Section 2, we first present 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 also show that the *most r-balanced min-cut bipartition* is NP-complete. We then present our heuristic algorithm for finding a min-net-cut r-balanced bipartition based on the repeated network flow technique in Section 3 with an efficient implementation. We compare our balanced bipartition results with that of the K&L heuristics and the spectral method in Section 4, and conclude the paper in Section 5.

2 Optimal Min-Net-Cut Bipartition

We first give some definitions in a flow network. An *s-t cut* (or *cut* for short) $(X, \bar{X})$ of a directed 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 (ending) node is in X and ending (starting) node is

*This work was partially supported by the Texas Advanced Research Program under Grant No. 003658459, by an Intel Foundation Graduate Fellowship, by a DAC Design Automation Scholarship, and by a grant from AT&T Bell Laboratories.

©1994 ACM 0-89791-690-5/94/0011/0050 $3.50

in $\bar{X}$ is called a *forward (backward) edge*. The *capacity of the cut* $(X, \bar{X})$ is the sum of the capacities on the *forward edges only* from X to $\bar{X}$. An *augmenting path* from u to v in G is a simple path from u to v in the undirected graph resulted from the network by ignoring edge directions that can be used to push additional flow from u to v. A *max-flow* f in G is a flow of maximum value $|f|$ from s to t.

Theorem 2.1 Max-flow min-cut theorem [FF62] *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 $\bar{X}$.*

2.1 Modeling a Net in a Flow Network

We model a *sequential circuit* as a digraph $N = (V, E)$, where V is the set of nodes representing gates and registers, and E is the set of edges representing the interconnections between gates and registers. Each node $v \in V$ is associated with a weight $w(v) \in R^+$. A *net* $n = (v; v_1, \ldots, v_l)$ is a set of outgoing edges from node v in N. For example in Figure 1, net a consists of two edges (r_1, g_1) and (r_1, g_2).

Given two nodes s and t in N, and an s-t cut $(X, \bar{X})$, the *net-cut* $net(X, \bar{X})$ of the cut is the set of nets in N that are incident to nodes both in X and in $\bar{X}$. A cut $(X, \bar{X})$ is a *min-net-cut* if $|net(X, \bar{X})|$ is minimum among all s-t cuts of N. In Figure 1, $net(X, \bar{X}) = \{b, e\}$, $net(Y, \bar{Y}) = \{c, a, b, e\}$, and $(X, \bar{X})$ is a min-net-cut,

In order to find a min-net-cut in N, 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 all cut edges 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 as follows (see Figures 2 & 3):

- 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 for all bridging edges and infinite capacity for 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, $w(u) = 0$.

Hence all nodes incident on net n are connected to n_1 and are connected from n_2 in N'. We show in Lemma 2.1 that the size of N' is only a constant factor larger than the size of N, for a connected graph N.

Lemma 2.1 *$|V'| \leq 3|V|$ and $|E'| \leq 2|E| + 3|V|$.*

Proof: The number of nets in N, denoted by $nets(N)$, is $\leq |V|$. V' contains all nodes in V, and two nodes for each net in N. Hence $|V'| = |V| + 2nets(N) \leq 3|V|$. For each net n 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 $|E'| = 2(|E| + nets(N)) + nets(N) = 2|E| + 3nets(N) \leq 2|E| + 3|V|$. □

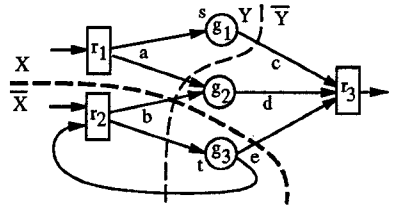

Figure 1: A digraph N representing a sequential circuit and its net-cuts.

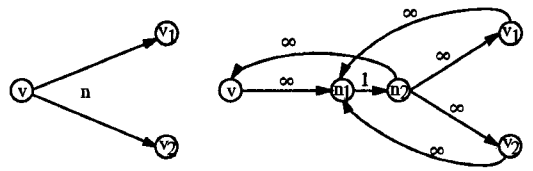

Figure 2: Modeling a net in N in the flow network N'.

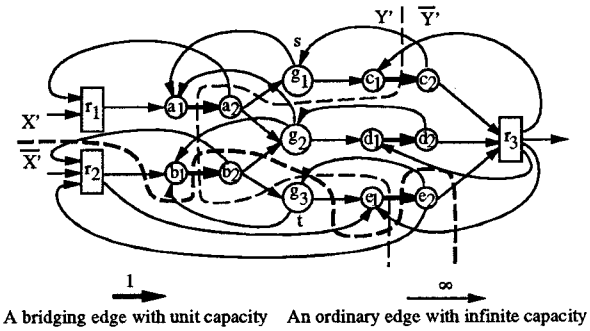

Figure 3: A flow network N' constructed from N.

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 [HM85] by modeling a net (or a hyperedge) as a star node, and then transforming a node-capacitated flow network into an edge-capacitated network [Law76] by splitting *every* node. The flow network obtained using this approach would require $4|V|$ nodes and $2|E| + 6|V|$ edges (see Figure 4 and compare with Figure 2). Our method is different from the above approach in that we split the nodes corresponding to the nets *only*, and hence use less nodes and edges in the resulting flow network. For a huge input circuit, our method translates into less memory

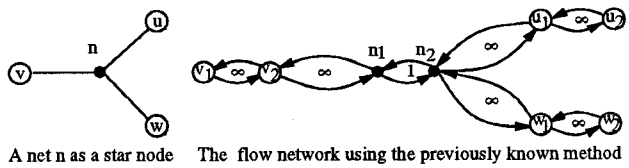

Figure 4: Modeling a net using the previously known method.

usage and faster runtime.

Another related result given in [IWW93] 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. Hence our method is able to exactly model a hypergraph without contradicting the result in [IWW93].

It is easy to see that N' is a strongly connected digraph. The strong connectivity of N' is the key to reduce the bi-directional min-net-cut problem to the minimum capacity cut problem that counts the capacity of the forward edges only. Lemma 2.2 and Corollary 2.2.1 show 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'.

Lemma 2.2
(1) If N has a cut $(X, \bar{X})$, then N' has a cut $(X', \bar{X}')$ of capacity $|net(X, \bar{X})|$.
(2) If N' has a cut $(X', \bar{X}')$ of capacity C, then N has a cut $(X, \bar{X})$ of net-cut size $|net(X, \bar{X})| \leq C$.

For example, in Figure 3, the cuts $(X', \bar{X}')$ and $(Y', \bar{Y}')$ are the corresponding cuts of $(X, \bar{X})$ and $(Y, \bar{Y})$ in Figure 1 respectively.

Corollary 2.2.1 *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 Lemma 2.2 (2). Then $(X, \bar{X})$ is a min-net-cut in N, and $|net(X, \bar{X})| = C$.*

2.2 Optimal Min-Net-Cut Algorithm

Based on Lemma 2.2 & Corollary 2.2.1, we give an optimal algorithm for finding a bipartition (not necessarily balanced) of a circuit $N = (V, E)$ with respect to two nodes s and t that minimizes the number of crossing nets.

Algorithm 1: Finding A Min-Net-Cut
0. Construct the flow network $N' = (V', E')$ for N as described in Subsection 2.1;
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 Lem. 2.2 (2).

Theorem 2.2 Algorithm 1 finds a min-net-cut in a circuit $N = (V, E)$, and it terminates in $O(|V||E|)$ time when N is a connected circuit.

Proof: The correctness of Algorithm 1 is established by Lemma 2.2 & Corollary 2.2.1.

Steps 0, 2-3 take time linear in the size of N. We use the simple augmenting path algorithm [FF62] to implement step 1. Finding an augmenting path in N' takes $O(|E'|)$ time. The number of augmenting paths in N' is no more than the number of bridging edges in the min-net-cut, which is at most $|V|$. Hence Algorithm 1 takes time $O(|V||E'|) = O(|V|(|E| + |V|)) = O(|V||E|)$ by Lemma 2.1. □

2.3 Most r-Balanced Min-Net-Cut Bipartition is NP-Complete

Min-net-cut bipartition may yield unbalanced components. The max-flow computation defines a set of min-net-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-net-cut that is the *most r-balanced* among all min-net-cuts defined by a max-flow, i.e., among all possible min-net-cuts $(X, \bar{X})$ defined by a max-flow find the min-net-cut such that $|w(X) - rW|$ is as close to 0 as possible, where W is the total weight.

Note that this problem is different from the known NP-Complete min-cut balanced bipartition problem in that the min-cut balanced bipartition fixes the weights of the two partitions and tries to minimize the number of crossing nets, while the *most r-balanced min-cut bipartition problem* requires that the number of crossing nets is minimum and tries to find a partition into X and $\bar{X}$ such that $w(X)$ is as close to rW as possible. We can show that the decision version of the problem is NP-complete by reducing the *weighted subset sum problem* [GJ79] to it.

3 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 partition 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 the terms of the original circuit and net-cuts, instead of the flow network constructed from the circuit (as shown in subsection 2.1) and forward bridging edges, when there is no confusion.

3.1 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. We allow the component weights to deviate from $(1 - \epsilon)rW$ to $(1 + \epsilon)rW$.

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 $v \in \bar{X}$ incident on a net in C;
   3.3. goto 1;
4. If $w(X) > (1 + \epsilon)rW$
   then 4.1. collapse all nodes in $\bar{X}$ to t;
   4.2. collapse to t a node $v \in X$ incident on a net in 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. This step is equivalent to increasing to infinity the capacities of all bridging edges reachable from s to v. The reasons why a more gradual method (i.e., increasing the unit capacities of the bridging edges by a fixed amount instead of infinity) is not adopted 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. A similar argument holds for step 4.2.

We now describe our strategy to pick a node in steps 3.2 & 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. 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.

3.2 Efficient Implementation

Iteratively applying Algorithm 1 in step 1 of Algorithm 2 to compute a max-flow and a min-net-cut from scratch can be very time-consuming. We show an efficient way to deal with the problem. In fact, it is not necessary to do the max-flow computation from scratch in every iteration. Instead, we only need to find additional flow to saturate the bridging edges of the net-cuts 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 the flow value along the augmenting path;
/* There is no more augmenting path from s to t.*/
2. Mark all nodes u s. t. $\exists$ 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 order in which the augmenting paths are found, Procedure 1 finds a max-flow with the same flow value as that computed in the collapsed flow network from scratch.

We show in Theorem 3.1 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 that of one max-flow computation.

Theorem 3.1 *If Procedure 1 is used to implement step 1 of Algorithm 2 after the first iteration, then Algorithms 2 has time complexity $O(|V||E|)$ for a connected circuit $N = (V, E)$.*

Proof: In the first iteration, step 1 of Algorithm 2 is implemented by Algorithm 1 and has time complexity $O(|V||E|)$ by Theorem 2.2. Since each augmenting path computation takes time $O(|E|)$, we prove that the total time complexity of step 1 of Algorithm 2 in the following iterations is still $O(|V||E|)$, 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 $|f| \leq |V|$ augmenting paths 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 3.1-3.2 or 4.1-4.2 of Algorithm 2. Hence the number of augmenting path computations in the following iterations is at most $2|V|$. □

Theorem 3.2 *The number of iterations and the final net-cut size of Algorithm 2 are non-increasing 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. The net-cut size found in each iteration is non-decreasing. □

Theorem 3.2 guarantees that with a larger ϵ deviation factor we can both improve the efficiency of Algorithm 2 and obtain a better partition solution.

4 Experimental Results

We implemented Algorithm 2 in a package called FBB using the C language, and integrated FBB in SIS/MISII [BRSV87]. Currently FBB only 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 36Mhz SS10 and 32MB memory (the C code was compiled with gcc without the optimizer). For each circuit, the number of gates and latches, the number of nets[1], and the average net degree (i.e., the average number of nodes connected to a net) are given in Tables 1&2.

Table 1 compares the average bipartition results of FBB with the results reported by Dasdan and Aykanat in [DA94]. The program SN is based on the K&L type heuristic algorithm in Sanchis [San89], which is a generalization of the Krishnamurthy [Kri84] algorithm. The program PFM3 is based on the K&L type heuristics with free moves as described in [DA94]. SN was

[1] Although we do not count a PI node in the second column in Tables 1&2, we do consider the set of nodes receiving a fanin from the same PI node as being in a net.

circuit				ave. net-cut			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 1: Comparison of bipartition results by SN, PFM3, and FBB (with $r = 1/2$ and $\epsilon = 0.1$).

circuit				best net-cut size			FBB improvem. % over		FBB
name	gates & latches	nets	ave. net degree	EIG1	PARABOLI	FBB	EIG1	PARABOLI	elapsed time
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	280.8
S38584	20859	20593	2.7	76	55	47	38.2	14.5	113.0
S38417	24033	23955	2.4	121	49	58	52.1	-18.4	273.6
Average							58.5	11.3	

Table 2: Comparison of bipartition results by EIG1, PARABOLI and FBB (with $r = 1/2$ and $\epsilon = 0.1$). All results allow up to 10% deviation from bisection.

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 1 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 1. 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 2 compares the best bipartition net-cut size of EIG1 (Hagen and Kahng [HK91]), PARABOLI (Riess, Doll, and Frank [RDF94]), and FBB. The results for EIG1 and PARABOLI were obtained from [RDF94]. The results for FBB were the best of 10 runs. The elapsed time[2] of FBB for the run that generates the best result was also recorded. All results in Table 2 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 partition result of FBB.

Note that different programs using the same MCNC benchmark circuits reported different properties such as 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 Table 3, we compare the average net-cut size and the average elapsed time of FBB for different values of ϵ. The data confirms our analysis in subsection 3.2 that both the runtime and the net-cut size produced by our algorithm are decreasing functions of ϵ. The elapsed time entries in Table 3 show that to the contrary of the common belief that network flow technique is slow and infeasible, our efficient implementation enables our network flow techniques based heuristic algorithm to partition large benchmark circuits with reasonable runtime. For example, the average elapsed time for bipartitioning the circuit S35932 of almost 20K gates is less than 20 minutes.

Hence we have shown that the deviation factor ϵ provides us with an effective way to directly control the tradeoff between improving the efficiency, the solution quality of FBB and relaxing the r-balanced criterion. Such kind of direct relation was not shown in the previous heuristics.

In the experiment, we have consistently found out

[2]The elapsed time of the execution of FBB, not the CPU time, was obtained by the *time* command in SIS/MISII.

circuit	ave. net-cut size			ave. elapsed time (sec.)		
$\epsilon =$	.1	.2	.4	.1	.2	.4
C1355	26.0	23.0	23.0	2.0	1.8	1.2
C2670	37.1	36.8	26.7	5.9	5.4	5.0
C3540	79.8	70.9	70.6	13.6	12.4	11.8
C7552	42.9	40.9	33.4	18.8	18.0	16.9
S838	14.7	12.9	9.0	1.2	1.1	1.1
S9234	112.4	93.3	74.5	69.4	55.0	52.7
S13207	113.7	100.7	100.3	121.8	102.8	99.58
S15850	96.0	90.1	76.0	122.6	111.8	96.9
S35932	213.3	177.3	165.1	1115.9	969.6	871.2

Table 3: Comparison of FBB results for different ϵ.

that for the runs with longer runtime (as compared with the average runtime), FBB always generates poorer solutions. This actually can be explained by Theorem 3.2 since the net-cut size is non-decreasing with more iterations. This property of FBB is in contrast to both the K&L type heuristics and the simulated annealing heuristic, 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 s and t. By doing so we will not lose any good solution, but we will further improve the efficiency of FBB.

5 Conclusions and Discussions

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

FBB has predictable behaviors in terms of the sizes of the two partitions, and the direct relation between efficiency, solution quality of FBB, and relaxing the r-balanced criterion by using a larger ϵ. Such kind of direct relation 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 has 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 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.

Acknowledgments We thank Andrew Kahng for kindly giving us pointers to the references.

References

[BRSV87] R. K. Brayton, R. Rudell, and A. L. Sangiovanni-Vincentelli. MIS: A Multiple-Level Logic Optimization. *IEEE Trans. on CAD*, pages 1061–1081, Nov. 1987.

[CHK92] J. Cong, L. Hagen, and A. Kahng. Net Partitions Yield Better Module Partitions. In *Proc. of the 29th ACM/IEEE Design Automation Conf.*, pages 47–52, 1992.

[DA94] A. Dasdan and C. Aykanat. Improved Multiple-Way Circuit Partitioning Algorithms. In *Int'l ACM/SIGDA Workshop on Field Programmable Gate Arrays*, Feb. 1994.

[Don88] W. E. Donath. *Logic Partitioning.* Preas and Lorenzetti eds., Benjamin/Cummings, 1988.

[Eve79] S. Even. *Graph Algorithms.* Computer Science Press, 1979.

[FF62] J. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[FM82] C. M. Fiduccia and R. M. Mattheyses. A Linear Time Heuristic for Improving Network Partitions. In *Proc. of the ACM/IEEE Design Automation Conf.*, pages 175–181, 1982.

[GJ79] M. Garey and D. S. Johnson. *Computers and intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[HK91] L. Hagen and A. B. Kahng. Fast Spectral Methods fo Ratio Cut Partitioning and Clustering. In *Proc. of the IEEE Int'l Conf. on Computer-Aided Design*, pages 10–13, Nov. 1991.

[HM85] T. C. Hu and K. Moerder. *Multiterminal Flows in a Hypergraph.* Hu and Kuh eds., IEEE Press, 1985.

[IWW93] E. Ihler, D. Wagner, and F. Wager. Modeling Hypergraphs by Graphs with the Same Min-Cut Properties. In *Info. Proc. Letters, 45*, pages 171–175, 1993.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, pages 671–680, May 1983.

[KL70] B. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning of Electrical Circuits. *Bell System Technical Journal*, pages 291–307, Feb. 1970.

[Kri84] B. Krishnamurthy. An Improved Min-Cut Algorithm for Partitioning VLSI networks. *IEEE Trans. on Computers*, pages 438–446, May 1984.

[Law76] E. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart & Winston, New York, 1976.

[RDF94] 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.*, pages 646–651, 1994.

[San89] L. A. Sanchis. Multiway Network Partitioning. *IEEE Trans. on Computers*, pages 62–81, Jan. 1989.