Project Proposal: Implementing Flow-based Balanced Bipartition Algorithm

Xiaojun Sun
Electrical and Computer Engineering
University of Utah
Salt Lake City, Utah 84112
Email: xiaojuns@ece.utah.edu

Abstract—**Flow-based balanced bipartition (FBB) algorithm is a max-flow/min-cut based circuit partitioning heuristic. The core algorithm is revised from max-flow algorithm on graphs/hypergraphs while taking the balancing criteria into consideration. Additionally, an incrementing max-flow algorithm is proposed to improve the complexity of whole approach. My project will focus on implementing FBB algorithm on variant netlists denoting by graphs and hypergraphs, exploring better initial assignments for attaining global optimal results and doing reasonable modification on original algorithms.**

I. Flow-based Bipartition Algorithm

A. Modeling a Net in a Flow Network

A net in circuit netlist could be represented as an edge or a hyperedge in graphs/hypergraphs. When we talk about the cost of cut set, it is defined as the number of nets in cut set. For general max-flow/min-cut problems, they deal with graphs with ordinary edges (2-ends edge). In order to utilize these high efficient max-flow/min-cut algorithms in our implementation, it is necessary to translate a net (hyperedge) into an ordinary edge, i.e. model the circuit in graphs which general max-flow/min-cut algorithm can deal with.

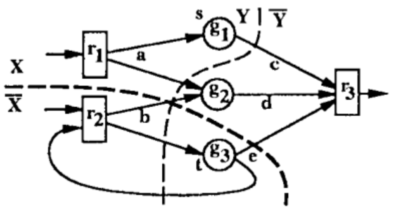

Fig. 1: Circuit nets example

Fig.1 is an example netlist with 2 different cut sets. In this figure, net *a* and *b* could be represented with hyperedges. Fig.2 explains a method to translate such nets into edges that could be dealt with general min-cut algorithm. 2 dummy vertices are inserted as long as a "bridging edge" with weight 1. Once this net is cut, the bridging edge must be included in the cut set from the output of any general min-cut algorithm, and cost is 1.

It showes from algorithm 1 that the size of newly constructed graph will not explode. And the following theorem guarantees that the result of general min-cut algorithm can be adopted.

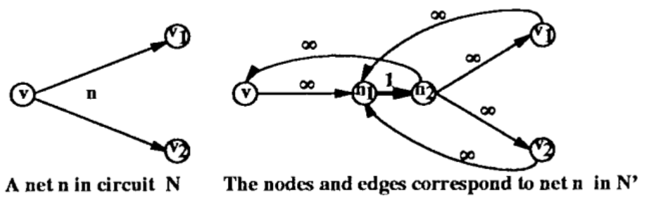

Fig. 2: Transform a Net to subgraph with bridging edge

Algorithm 1 Algorithm for constructing flow network from circuit netlists

```
 1: function CONSTRUCT(Netlist allowing hyperedges $N = (V,E)$)
 2:     $V' \leftarrow V$ /*copy all vertices*/
 3:     for each net $n \in E$ do
 4:         Insert auxiliary vertices $n_1$ and $n_2$ into $V'$
 5:         Insert bridging edge $e = (n_1, n_2)$ with unit weight into $E'$
 6:         for each node $u$ incident to $n$ do
 7:             Insert edge $(u, n_1)$ and $(n_2, u)$ with infinite weight into $E'$
 8:         end for
 9:     end for
10:     return graph $N' = (V', E')$
11: end function
```

Theorem 1.1: N has a cut of net whose size is at most C if and only if N' has a cut of capacity at most C.

B. Min-cut Balanced Bipartition

It is easy to see a minimum cut of nets in netlist can be computed from general min-cut algorithm, based on our netlist-graph translation. However, the min-cut cannot guarantee a balanced bipartition. It is necessary to build an algorithm iterating with optimal min-cut and improve the balance every loop, which is the FBB algorithm we want to propose.

Fig.3 illustrates what algorithm 2 does. First step, s and t are randomly chosen from all vertices, and a min-cut $(X1, \overline{X1})$ of cost 2 is computed. Note that a small dot denote a pair of auxiliary nodes we added for each net. Apparently this is not a balanced cut partition, so we merge X ($\varnothing$ in first iteration)

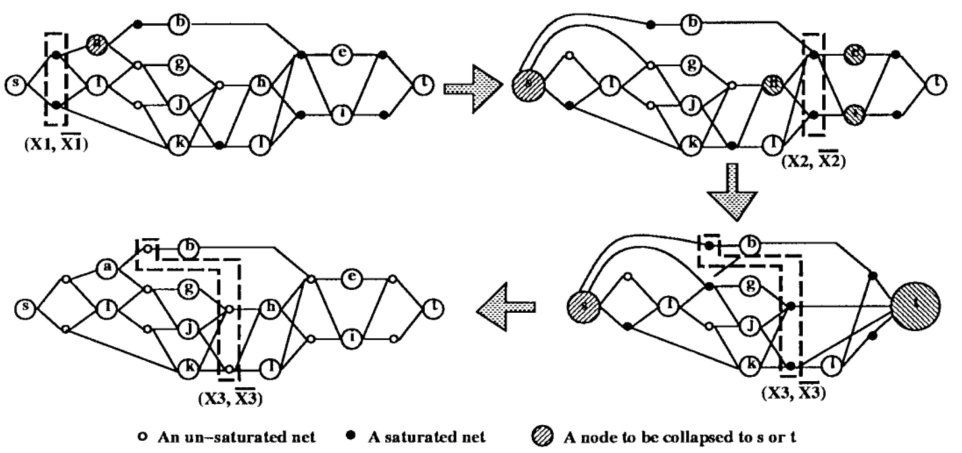

Fig. 3: An example on FBB algorithm execution

Algorithm 2 FBB Algorithm

```
 1: function FBB(graph $G = (V,E)$)
 2:     Randomly pick a pair of vertices as source $s$ and sink
        $t$, delete all edges starting from $t$ or ending at $s$
 3:     $C \leftarrow$ MINCUT($G$)
 4:     $X \leftarrow$ vertices from source part $\setminus C$
 5:     $X' \leftarrow$ vertices from sink part $\setminus C$
 6:     while $C$ violates balancing criteria do
 7:         if $X$ is too small then
 8:             Collapse $X$ to $s$
 9:             Collapse to $s$ a vertex $v \in X'$ adjacent to $C$
10:         else
11:             Collapse $X'$ to $t$
12:             Collapse to $t$ a vertex $v \in X$ adjacent to $C$
13:         end if
14:         $C \leftarrow$ MINCUT($G$)
15:         $X \leftarrow$ vertices from source part $\setminus C$
16:         $X' \leftarrow$ vertices from sink part $\setminus C$
17:     end while
18:     return $C$
19: end function
```

Algorithm 3 Incremental Flow Min-cut

```
 1: function INCFLOWMINCUT(residue flow graph $G'$)
 2:     copy flow info from last iteration
 3:     while $\exists$ an augmenting path from $s$ to $t$ do
 4:         saturate this path
 5:     end while     ▷ /* there is no more augmenting path
        from $s$ to $t$ */
 6:     for all vertices $u$ do
 7:         if $\exists$ an augmenting path from $s$ to $u$ then
 8:             Insert $u$ to set $X$
 9:         end if
10:     end for
11:     for all vertices $u \in X$ do
12:         if $e = (u,v)$ is bridging edge and $v \notin X$ then
13:             Insert $e$ into $C'$
14:         end if
15:     end for
16:     return $C$ whose nets contain $C'$ as min-cut
17:     return $X$, and $V' \setminus (X \cup C')$ as $X'$
18: end function
```

into s, then pick vertex a from X', also merge it into s. In the second iteration, we run min-cut algorithm again, and get a cut $(X2, \overline{X2})$. The cost is also 2, but this is still not balanced in the other direction. This time we merge X' (vertices e and i) into t, as well as vertex h chosen from X. In third iteration, a cut $(X3, \overline{X3})$ of cost 3 satisfies the balancing criteria, which is returned as final result.

C. *Efficient Min-cut Computation*

The algorithm's complexity is retarded by computing min-cut within each iteration. In order to improve this, a new incremental flow computation based algorithm is proposed. The main idea is to keep track of the flow information from last iteration, and start the max-flow computing from this initial flow instead of zero flow.

The refined algorithm with k iterations is proved to have same complexity as a zero-flow min-cut algorithm.

II. EXPERIMENT

I plan to execute experiment on ISCAS benchmarks, and do comparison with K-L based tools and *hMETiS* tool. This heuristic depends on the initial assignment of source and sink, so FBB should be executed for multiple times to test average performance.

REFERENCES

[1] Yang H H, Wong D F. *Efficient network flow based min-cut balanced partitioning*[M], The Best of ICCAD. Springer US, 2003: 521-534.