Minimum Cut Algorithm for Hypergraph

Razmik Alaverdyan

Abstract - **The algorithm for computing global minimum cut in hypergraph is presented. This algorithm is a non-flow based algorithm. Since the netlist of a circuit can be modeled naturally as a hypergraph, this opens the opportunity for finding very high quality solutions for the circuit partitioning problem.**

Keywords – **Hypergraph, circuit partitioning, minimum cut, algorithm, tightly connected**

I. INTRODUCTION

Partitioning plays a central role in VLSI system design [1]. Partitioning tools are required for dividing system into smaller and more manageable subsystems. A good partitioning should limit the number of signal nets between different subsystems to ensure high system performance, because signal delays are different between same and the other subsystems. So min-cut partitioning that minimizes the number of interconnections between different subsystems is desired.

A system can be partitioned into smaller components by recursively dividing each big component into two components. Thus solving the two-way partitioning problem provides the basis for solving the general partitioning problem. In this paper, we focus our attention on the two-way partitioning problem.

As a net can connect multiple modules, the natural representation for a circuit netlist is a hypergraph where the nodes correspond to the modules and the edges correspond to the interconnections. And the cost of each edge reflects the cost of the corresponding interconnection. There are many algorithms for solving the minimum cut problem for graphs known in the literature [1]. Most of these algorithms rely on maximum flow computations motivated by the max-flow min-cut theorem (the max-flow min-cut theorem states the dual relationship between a maximum flow between two nodes s and t and a minimum $s-t$ cut that separates s and t) by Ford and Fulkerson [2]. Several researchers [3, 4, 5] presented some algorithms for computing minimum cut without using any flow computation. The fastest algorithms known today are by Nagamochi and Ibaraki [4], and Stoer and Wagner [5]. Their algorithms have a running time of $O(mn+n^2 \log n)$ where m is the number of edges and n is the number of nodes in the graph. These algorithms are designed for finding minimum cuts in graphs but not in hypergraphs. Though some researchers have tried to find ways to model a hypergraph with a graph, Ihler et al. [6] proved that there is no perfect transformation to model hypergraphs by graphs with the same mincut properties. That means any strategy that relies on hypergraph to graph transformation and the use of a minimum cut algorithm for graphs can at best find approximate solutions to an optimal partition of the hypergraph.

R. Alaverdyan is with the Department of Support, Synopsys Armenia CJSC – Yerevan, 41 Arshakunyats ave., 0026 Yerevan, Armenia, e-mail: arazmik@mail.ru

The hypergraph minimum cut problem was considered in [7, 8]. Hu and Moreder first considered the problem in [7]. They modeled each net x as a star node (Figure 1) in a node-capacitated flow network where each star node has a capacity equals to the cost of the corresponding net and all other nodes have infinite capacity. They presented a simple flow augmenting type algorithm to compute the minimum node separator of the network. Since any non-star node has infinite capacity, the minimum node separator cannot contain any non-star node. Thus a minimum node separator must be a subset of the star nodes which corresponds to the set of nets in a minimum cut of the hypergraph.

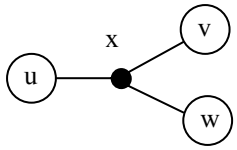

Figure 1: A net x modeled as a star node connecting all modules in the net.

However, the node-capacitated flow network can be readily transformed into an arc-capacitated network using a technique due to Lawler [9]. Figure 2(a) shows how the star in Figure 1 is transformed. After transforming to an arc-capacitated network, more efficient flow algorithms can be applied. Yang and Wong [8] gave a more compact transformation (Figure 2(b)) that uses less nodes and edges.

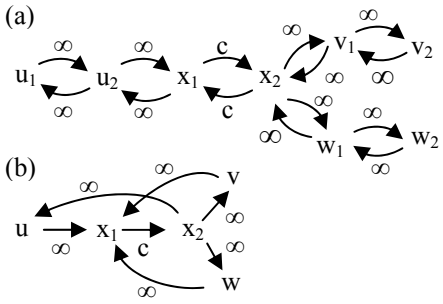

Figure 2: (a) Transforming a star (c is the cost of star node x). (b) A more compact transformation.

The fastest known algorithms for computing a minimum $s-t$ cut that separates two fixed nodes in a network are flow-based algorithms which takes $\Omega(n',m')$ time where n' and m' are the numbers of nodes and arcs in the network. Using the transformation of [8], the network constructed for computing a hypergraph minimum cut has $n+2m$ nodes and $m+2p$ arcs where m, n and p are the numbers of edges, nodes, and terminals in the hypergraph. Computing a hypergraph minimum $s-t$ cut using the flow-based approach takes $\Omega((n+2m)(m+2p)$ time. Note that a global minimum cut can be found by computing $n-1$ minimum $s-t$ cuts.

In this paper we present an algorithm which is not flow based and it computes a global minimum cut in a hypergraph. The algorithm runs in $O(np+n^2 \log n)$ time and it is a fast hypergraph minimum cut algorithm. Note

that the number of edges m can be exponential in the number of nodes n in a hypergraph, so computing a global minimum cut using our algorithm is even faster than just computing one minimum $s-t$ cut by the flow-based approach which takes $\Omega(mp)$ time. Our algorithm is a non-trivial extension of the result by Stoer and Wagner [5] which works for graphs only.

II. PRELIMINARIES

A hypergraph $H=(V,E)$ is defined by its node set V and edge set E. While the edges of a graph connect exactly two nodes each, the edges of a hypergraph can connect two or more nodes each (Figure 3). The nodes connected by an edge e are called the terminals of e.

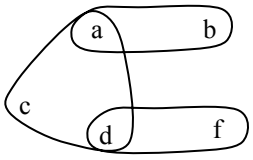

Figure 3: A hypergraph.

A cut $C=(X,\overline{X})$ is a partition of the set of nodes into two non-empty sets X and $\overline{X}=V-X$. The size of a cut $C=(X,\overline{X})$ is the total cost of all edges that have terminals in both X and $\overline{X}$, and is denoted by $\omega(C)$. An edge that has terminals in both X and $\overline{X}$ is called a cut edge. For example, $C=(\{a,c\},\{b,d,f\})$ is a cut of the hypergraph in Figure 4(a), and $\omega(C)=1+2=3$.

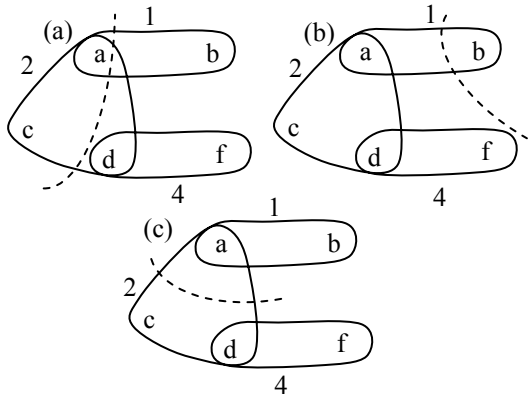

Figure 4: (a) A cut. (b) A minimum cut. (c) A minimum *s*-*t* cut for *s* = *a* and *t* = *c*

A global minimum cut or simply minimum cut of a hypergraph H is a cut of H that has the smallest cut size among all the possible cuts of H. A minimum $s-t$ cut is a cut among the set of cuts that separate nodes s and t (i.e., $s\in X$ and, $t\in\overline{X}$, or vice versa) with the smallest cut size. Referring to Figure 4(b), cut $(\{b\},\{a,c,d,f\})$ is a global minimum cut and has a size of 1. Referring to Figure 4(c), cut $(\{a,b\},\{c,d,f\})$ is a minimum $s-t$ cut for $s=a$ and $t=c$, and has a size of 2.

III. THE ALGORITHM

The algorithm for computing a minimum cut of a hypergraph is based on the following observation. In case of having procedure P that can compute a minimum $s-t$ cut for some nodes s and t quickly for any hypergraph, a global minimum cut can be computed quickly by using the procedure $n-1$ times where n is the number of nodes in the hypergraph.

A global minimum cut can be found for a hypergraph with two nodes by applying procedure P once because a minimum $s-t$ cut is also a global minimum cut for a hypergraph with only two nodes. If a global minimum cut for a hypergraph with k nodes is computed, a global minimum cut for a hypergraph H with $k+1$ nodes can be computed using one more application of procedure P. Because the smaller of a minimum $s-t$ cut of H and a minimum cut of H with nodes s and t on the same side must be a global minimum cut of H (recall that a minimum $s-t$ cut is a minimum cut with nodes s and t on opposite sides). So, a global minimum cut for hypergraph with $k+1$ nodes can be computed by finding a minimum $s-t$ cut using procedure P, then computing a global minimum cut of the reduced hypergraph with nodes s and t merged, and finally picking the smaller of the two.

So, the main question is: how can a minimum $s-t$ cut for some nodes s and t be computed quickly for any hypergraph? Note that the choice of s and t that are easy to compute a minimum $s-t$ cut efficiently is free. It is shown in [5] how a minimum $s-t$ cut for some nodes s and t can be computed quickly in a graph, here it is shown how it can be done in a hypergraph.

A node v is tightly connected to a node subset A if there is an edge that has node v as a terminal and has all other terminals in A, but no terminals in $V/(A\cup\{v\})$. For example, for the hypergraph in Figure 5, node d is tightly connected to set $\{a,b,c\}$ because of edge $\{d,a,b\}$, and node g is tightly connected to set $\{a,b,c\}$ because of edge $\{g,c\}$. But node f is not tightly connected to set $\{a,b,c\}$ (note that edge $\{f,c,g\}$ does not make f tightly connected to set $\{a,b,c\}$). For any $A\subset V$, $\omega(A,v)$ is defined to be the sum of the weights of those edges that have node v as one of the terminals and have all other terminals in A, but no terminals in $V/(A\cup\{v\})$. (Note that $\omega(A,v)$ is zero if v is not tightly connected to A.) The node most tightly connected to a node subset A is defined as a node $v\notin A$ such that $\omega(A,v)$ is maximum. The major difference between our hypergraph minimum cut algorithm and the graph minimum cut algorithm in [5] is in the definitions of $\omega(A,v)$ and the node most tightly connected to a node subset A. The following algorithm must find a minimum $s-t$ cut for any hypergraph.

Some terminologies needed to prove that our algorithm correctly finds a minimum $s-t$ cut in the input hypergraph H are introduced. A sequence $\{a_i\}$ is defined from the set of nodes in H according to the order they are added to A by the algorithm, i.e., $a_i=$ the i-th node added to A. Hence, $s=a_{n-1}$ and $t=a_n$ where n is the number of nodes in H. $pred(v)$ is used to denote the

node immediately preceding node v in the sequence. And A_v is used to denote the set of nodes in the prefix sequence ($a_1, a_2, a_3, ..., pred(v)$).

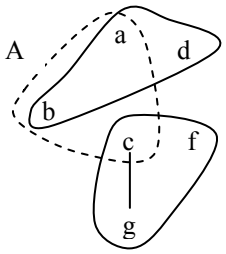

Figure 5: Both nodes *d* and *g* are tightly connected to *A={a,b,c}* but node *f* is not.

Fast Minimum $s-t$ Cut Algorithm

Input: A Hypergraph H.

Output: A minimum $s-t$ cut C^* of H and the choice of s and t.

$x :=$ an arbitrary node in H;

$A := \{x\}$;

while $A \neq V$ do

$v :=$ node in V/A most tightly connected to A (i.e. $\omega(A,v)$ is maximum);

$A := A \cup \{v\}$;

od; /* $A = V$ */

$s :=$ 2nd last node added to A;

$t :=$ last node added to A;

$C^* := (A/\{t\}, t)$;

Given a $s-t$ cut C, v is an active node if nodes v and $pred(v)$ are on opposite sides of the cut C. Let C_v denote the set of edges whose terminals are all in $A_v \cup \{v\}$ that are cut by C.

It is clear that the computed cut $C^* := (V/\{t\}, t)$ is a $s-t$ cut of hypergraph H. It remains to show that $\omega(V/\{t\}, t) \leq \omega(C)$ for any $s-t$ cut C of H where $s = a_{n-1}$ and $t = a_n$.

Consider an arbitrary $s-t$ cut C of H where $s = a_{n-1}$ and $t = a_n$, we want to show that $\omega(V/\{t\}, t) \leq \omega(C)$. Since $A_t = V/\{t\}$ and $C_t = C$, it is equivalent to prove that $\omega(A_t, t) \leq \omega(C_t)$. We will prove a more general result in Lemma 1, which says that $\omega(A_v, v) \leq \omega(C_v)$ for any active node v. Note that t is an active node because $prev(v) = s$, and by assumption C is a $s-t$ cut, so s and t are on opposite sides of C. Thus we are done if we can prove the lemma.

Lemma 1: For any $s-t$ cut C of hypergraph H where $s = a_{n-1}$ and $t = a_n$, we have $\omega(A_v, v) \leq \omega(C_v)$ for every active node v in the sequence $\{a_i\}$.

Proof: Lemma will be proved by induction. Suppose we have a fixed $s-t$ cut C where $s = a_{n-1}$ and $t = a_n$.

Base case: If v is the first active node, then all nodes in A_v must be on one side of the cut C, and v must be on the other side. So, any edge that has v as one of its terminals and has all other terminals in A_v must be cut by C and hence in C_v. Thus we have $\omega(A_v, v) \leq \omega(C_v)$.

Inductive step: Suppose that the statement holds for active node u, and v is the next active node after u. We will show that the statement also holds for active node v. By the definition of $\omega(A_v, v)$, it can be written as the sum of these terms.

$$\omega(A_v, v) = \omega(A_u, v) + \omega(A_v / A_u, v) + \omega(A_u, A_v / A_u, v) \quad (1)$$

where $\omega(A_u, A_v / A_u, v)$ is the total cost of all edges with v as a terminal that also have terminals in both A_u and A_v / A_u, but not in $V/(A_v \cup \{v\}$. As u is the node most tightly connected to A_u, we have $\omega(A_u, v) \leq \omega(A_u, u)$. And by the induction hypothesis, $\omega(A_u, u) \leq \omega(C_u)$. So,

$$\omega(A_v, v) = \omega(A_u, v) + \omega(A_v / A_u, v) + \omega(A_u, A_v / A_u, v) \leq$$
$$\leq \omega(C_u) + \omega(A_v / A_u, v) + \omega(A_u, A_v, v) \quad (2)$$

Because all nodes between u and v in the sequence are non-active nodes by the assumption, all nodes in A_v / A_u must be on one side of the C, while v must be on the other side as it is active. So any edge with v as a terminal and have at least one terminal in A_v / A_u must be cut by C. Hence, any edge counted in $\omega(A_v / A_u, v)$ or $\omega(A_u, A_v / A_u, v)$ must belong to C_v. And as C_u is a subset of C_v, any edge counted in $\omega(C_u)$ must belong to C_v. Finally, notice that the set of edges counted in $\omega(C_u)$, the set of edges counted in $\omega(A_v / A_u, v)$, and the set of edges counted in $\omega(A_u, A_v / A_u, v)$ are mutually disjoint by their definitions. So, we have $\omega(A_u, v) \leq \omega(C_u) + \omega(A_v / A_u, v) + \omega(A_u, A_v / A_u, v) \leq$ $\leq \omega(C_u)$. From the above, we have $\omega(A_u, v) \leq \omega(C_v)$ for any active node v with respect to the cut C. As C can be any arbitrary $s-t$ cut of H, the lemma is proved.

We will illustrate how the fast minimum $s-t$ cut algorithm works with an example. Consider the hypergraph in Figure 6. Suppose node a is picked as the arbitrary node to put into A first. Then A becomes $\{a\}$, and the node most tightly connected to A is b since $\omega(\{a\}, b) = 2$ while $\omega(\{a\}, e) = 1$ and $\omega(\{a\}, v) = 0$ for any other node v. So the second node added to A is b and A becomes $\{a,b\}$. Now, there are two nodes most tightly connected to A, namely, nodes c and d, since $\omega(\{a,b\}, c) = \omega(\{a,b\}, d) = 3, \omega(\{a,b\}, e) = 1$, and $\omega(\{a,b\}, f) = 0$. Suppose we choose to add node c into A, then A becomes $\{a,b,c\}$. It can be checked that d is the node most tightly connected to $\{a,b,c\}$, hence node d is added to A next, followed by node e, and finally node

f. So, we get $C^* = (\{a,b,c,d,e\},\{f\})$, $s = e$ and $t = f$. The cut size of C^* is 3. It can be checked that C^* is indeed a minimum $s-t$ cut for $s = e$ and $t = f$.

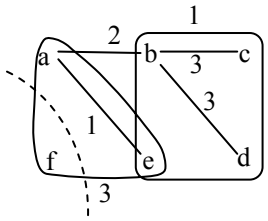

Figure 6: A hypergraph.

The global minimum cut of the hypergraph in Figure 6 is found by five minimum $s-t$ cut computations. Figure 7(a) shows the cut computed by the fast minimum $s-t$ cut algorithm assuming the nodes are added to A in the order a,b,c,d,e,f. The cut size is 3 and the cut separates e and f. So nodes e and f are merged before the second minimum $s-t$ cut computation. Figure 7(b) shows the cut computed by the fast minimum $s-t$ cut algorithm assuming the nodes are added to A in the order ef,a,b,d,c. The cut size is 4 and the cut separates d and c. So nodes d and c are merged before the third minimum $s-t$ cut computation. Figure 7(c) shows the cut computed by the fast minimum $s-t$ cut algorithm assuming the nodes are added to A in the order cd,b,a,ef. The cut size is 7 and the cut separates a and ef. So nodes a and ef are merged before the fourth minimum $s-t$ cut computation. Figure 7(d) shows the cut computed by the fast minimum $s-t$ cut algorithm assuming the nodes are added to A in the order aef,b,cd. The cut size is 7 and the cut separates b and cd. So nodes b and cd are merged before the fifth minimum $s-t$ cut computation. Figure 7(e) shows the cut. And the cut size is 3. The minimum of the five cuts computed above is a global minimum cut. In this example, the first and the fifth computed cuts are two different global minimum cuts.

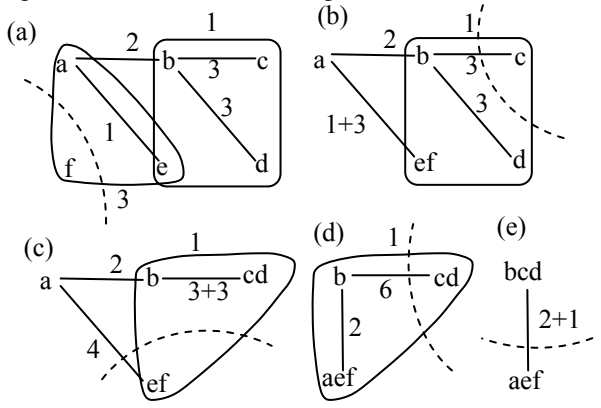

Figure 7: Computing a global minimum cut.

IV. Implementation and Complexity

Let n be the number of nodes in hypergraph H. Let p be the total number of terminals in all the edges of H. The fast minimum $s-t$ cut algorithm can be implemented to run in $O(p+n\log n)$ time. In the fast minimum $s-t$ cut algorithm, we need to keep track of the value of $\omega(A,v)$ for all node v not in the current set A. And we have to find the node v not in the current A with the maximum $\omega(A,v)$ value quickly. We can put the nodes into a priority queue using the value of $\omega(A,v)$ as the key of node v. When we add a node v to A, we will remove it from the priority queue, and update $\omega(A,u)$ for any node u that becomes the last terminal of an edge to remain outside of A (i.e., u and v are terminals of the same edge e and they are the last two terminals of e yet to put into A). We can identify such last node of an edge by keeping a counter for each edge. The value of the counter of an edge e is set to the number of terminals of the edge initially, and is decremented by 1 each time a terminal of e is added to A. Every time the counter for an edge e becomes 1, we scan all the terminals of e to find the last remaining node u outside A and increase the value of $\omega(A,u)$ by $\omega(e)$. During the whole algorithm the total time spent for decrementing the counters and scanning when a counter becomes 1 is $O(p)$. And there are m IncreaseKey operations where m is the number of edges in H, and n ExtractMax operation. If we implement the priority queue using Fibonacci heaps, the amortized times for performing IncreaseKey and ExtractMax are $O(1)$ and $O(\log n)$, respectively. Hence, the total running time of the fast minimum $s-t$ cut algorithm is $O(p+m+n\log n) = O(p+n\log n)$.

We can compute a global minimum cut in a hypergraph in $O(np+n^2\log n)$ time by applying the fast minimum $s-t$ cut algorithm $n-1$ times. For VLSI circuits, usually $p = cn$ for some c around 3 and 4. So the time becomes $O(n^2\log n)$.

V. Conclusion

Because of the importance of the hypergraph minimum cut problem in finding good partitions for VLSI circuits, many researchers have studied this problem. Previously, the most efficient approach to find a minimum cut in a hypergraph is to transform the hypergraph into a flow network and then apply a flow-based algorithm on the network. However, in this paper we presented a fast non-flow based approach to compute a minimum cut in the hypergraph directly. Our algorithm is a fast hypergraph minimum cut algorithm.

Finding a global minimum cut in a hypergraph is an important step towards finding better solutions for the complicated system partitioning problem. If the minimum cut is not balanced, we can use a repeated cut process as in [8] to grow or shrink a partition to the desired size without increasing the cut value too much. Yang and Wong [8] have shown that this technique works very well for finding balanced $s-t$ cuts in practice, so one can only expect to find better partitions when it is applied together with our

algorithm since we do not require the use of a fixed pairs of nodes s and t.

REFERENCES

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications,* Prentice Hall, 1993.
[2] J.R. Ford and D.R. Fulkerson, *Flows in Networks,* Princeton University Press, 1962.
[3] D.R. Karger and C. Stein, *A New Approach to the Minimum Cut Problem*, in Journal of the ACM, July 1996, Vol. 43, No 4, pp. 601- 640.
[4] H. Nagamochi and T. Ibaraki, *Computing Edge-Connectivity in Multigraphs and Capacitated Graphs*, in SIAM Journal of Discrete Mathematics, 1992, Vol. 5, No 1, pp. 54-66.
[5] M. Stoer and F. Wagner, *A Simple Min-Cut Algorithm*, in Journal of the ACM, July 1997, Vol. 44, No 4, pp. 585-591.
[6] E. Ihler, D. Wagner, and F. Wagner, *Modeling Hypergraphs by Graphs with the Same Mincut Properties*, in Information Processing Letters, March 1993, Vol. 45, No 4, pp. 171-175.
[7] T.C. Hu and K. Moerder, *Multiterminal Flows in a Hypergraph*, in VLSI Circuit Layout: Theory and Design, IEEE Press, pp. 87- 93, 1985.
[8] H. Yang and D.F. Wong, *Efficient Network Flow based Min-Cut Balanced Partitioning*, in Proc. of the IEEE/ACM Int'l Conf. on Computer-Aided Design, pp. 50-55, 1994.
[9] E.L. Lawler, *Combinatorial Optimization: Networks and Matroids,* Hot, Rinehart, &Winston, New York, 1976.
[10] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout,* Wiley-Teubner, 1990.
[11] B. Preas and M. Lorenzetti, *Physical Design Automation of VLSI Systems*, Benjamin/Cummings, 1988.
[12] C.J. Alpert and A.B. Kahng, *Recent Directions in Netlist Partitioning: A Survey*, in Integration: the VLSI Journal, 1995, Vol. 19, No 1-2, pp. 1-81.
[13] R. Gomory and T.C. Hu, *Multi-Terminal Network Flows*, in Journal of SIAM, 1961, Vol. 9, No 41961, pp. 551-570.