Finding bipartition respecting natural dense clusters

ABHIJIT S. DESHPANDE
Department of Computer Sci. and Engg.
Indian Institute of Technology, Bombay
Mumbai 400 076
INDIA

SACHIN B. PATKAR
Department of Mathematics
Indian Institute of Technology, Bombay
Mumbai 400 076
INDIA

H. NARAYANAN
Department of Electrical Engg.
Indian Institute of Technology, Bombay
Mumbai 400 076
INDIA

Abstract: - We present an efficient heuristic for finding good bipartitioning of the vertex set of a hypergraph, which respects the *dense clusters* inside. In a hypergraph modeling a VLSI netlist, these *dense clusters* typically correspond to dense combinational blocks and are better kept intact for some obvious VLSI design benefits. Our approach to identify the *dense clusters* is based on the theory of submodular functions. Once these clusters are identified, we make use of FM algorithm to get a good bipartition, providing some hints to FM to keep these clusters intact. This approach not only respects the natural clusters, but also produces bipartitions with better net-cut in quite a few cases. Hence, we propose that this approach can be used as an alternate strategy to find a good bipartition, which respects the *dense clusters*.

Key-Words: - Circuit Partitioning, Dense Clusters, Submodular Functions.

1 Introduction

VLSI circuit partitioning is a vital part of *physical design* phase of the design process. The essence of circuit partitioning is to divide a circuit into a number of sub-circuits such that size of each sub-circuit is less than some prescribed limit and the complexity of interconnections among these sub-circuits is minimized. Since, the typical sizes of hypergraphs arising in VLSI domain are quite large, we need to have an algorithm, which is quite efficient in the sense of execution time. So far, an algorithm proposed by Fidducia and Mattheyses [1] is the best-known algorithm, which runs almost linearly in the size of netlist. But FM algorithm works purely on the objective of minimizing the number of interconnects, and lacks the support to treat *dense clusters* in special manner.

In this paper, we describe a new practical and efficient approach that aims at finding a good bipartition, which respects the natural *dense clusters* inside the netlist. These clusters typically represent the dense combinational blocks. Our approach is based on the theory of submodular functions [2]. We have tested this approach on ISDP98 IBM benchmarks and found to be producing bipartitions with better net-cut in atleast 50% of the test cases. So, it is evident that this approach not only respects the natural clusters, but also finds better bipartition in the process.

To identify *dense clusters*, we use a network-flow model, which is quite similar to one suggested by Huang and Kahng [3]. They aim to find a *k*-way partitioning, whose blocks have maximum sum of densities and use this model to identify such dense blocks. But our approach differs from it in the sense that our ideas are based on the concept of *fusion set* (defined in the

next section), obtained by minimization of submodular function. And we use this network-flow model for this minimization. Also, the approach suggested by Huang et al. is computationally expensive, and has been tested only on small benchmarks. On the contrary, as explained later during analysis, our approach does not take much time and has been tested on large ISPD98 benchmarks of size upto 210K cells.

We use a hypergraph model of netlist, which is a more natural way to represent it. A circuit, represented as netlist, is essentially a collection of cells and nets, where each net connects two or more cells. Similarly hypergraph is also a collection of vertices and hyperedges, where each hyperedge can connect to two or more vertices. Hence vertices can model cells, whereas hyperedges can model nets.

The paper is organized as follows. Section 2 describes relevant notations and definitions. In section 3, we describe the core idea of identifying *dense clusters*, which is based on the idea of *fusion sets* and related theory of submodular functions. Section 4 describes our heuristic approach of incorporating the idea of identifying *dense clusters* in bipartitioning using FM algorithm. Section 5 presents the results of the experiments carried out on ISPD98 benchmarks of sizes ranging from 12K to 210K cells.

2 Preliminaries and notation

We use the notation $H(V, E_h)$ to represent the hypergraph, with vertex (cell) set V, hyperedge (net) set E_h and positive weight assignments $w_v(.)$ and $w_e(.)$ on vertex set and hyperedge set respectively.

A function $f: 2^v \to \Re$ is submodular iff $f(X)+f(Y) \geq f(X \cup Y)+f(X \cap Y)$, $\forall X, Y \subseteq V$. Now, we define a submodular function $-|\tilde{E}(.)|: 2^v \to \Re$ (refer [2] for proof of its submodularity), where $\tilde{E}(U)$ denotes the set of edges which are completely contained in U. The function $-w_e(\tilde{E}(.)): 2^v \to \Re$, which is a composition of $w_e(.)$ and $\tilde{E}(.)$, is also a submodular function as $w_e \geq 0$.

2.1 Partition associate function

Given a sub-modular function $f(X)$, we define its partition associate function $\overline{f}(\Pi)$ as $\overline{f}(\Pi) = \sum_{v_i \in \Pi} f(V_i)$.

2.2 Zero Singleton Submodular function

Let $f: 2^s \to \Re$ be a submodular function. Then the corresponding *zero singleton submodular* (z.s.s.) can be given as $\hat{f}(U) = f(U) - \sum_{u \in U} f(\{u\})$. A z.s.s function $\hat{f}$ has following properties:

- $\hat{f}$ is submodular
- $\hat{f}(e) = 0, \forall e \in S$

2.3 Fusion Set

If $f(.)$ is a z.s.s. on subsets of *S*, then a set T ⊆ S is called as a *fusion set* of f iff:

- $f(T) < 0$,
- $f(T) \leq f(R), \forall R \subseteq T$, and
- All subsets of T, on which $f(.)$ achieves a negative value, have a common element.

Now, we state an important theorem, which is the core idea behind finding the *dense clusters*.

Theorem: Let $f(.)$ be a *z.s.s.* function on subsets of *S*, and *N* be a *fusion set* of $f(.)$. Then there exists a partition Π of *S* such that $\overline{f}(.)$ reaches a minimum on it and *N* is contained in one of the *blocks* of Π.

For proof of the above theorem and details on *partition associate function*, *zero singleton submodular functions* and *fusion sets*, the reader is referred to [2].

3 Submodular functions based approach

As implied by the theorem stated in the previous section, if N is a *fusion set* of $f(.)$, then it is always beneficial to keep it intact (uncut) during the process of partitioning.

The algorithm (refer [2] for details) to identify the *fusion set* requires O($|V|$) minimizations of corresponding submodular function, which means that it is computationally expensive. So, we adapt a more practical approach of identifying a *dense set*. The subset $U_0 \subseteq S$, is called as *dense set*, if it has following property:

- $f(U_0) = f(Fusion\ Set)$.

And such a set $U_0 \subseteq S$, is identified by finding out a subset, which minimizes the given *z.s.s.* function *f(.)* over all subsets of S.

Now consider the submodular function, $f(U) = -w_e(\tilde{\mathrm{E}}(U)) - k$. It accounts for the weight of edges, which lie completely inside subset U. Note that there is a bias by "-k", which is explained later. The partition associate of this function is given by $\overline{f}(\Pi) = -\sum_{V_i \in \Pi} w_e(\tilde{\mathrm{E}}(V_i)) - k * |\Pi|$. It is obvious that the partition minimizing $\overline{f}(\Pi)$, maximizes the number of edges inside the blocks of the partition, and hence reduces the net-cut. The "-k*|Π|" part of the function makes sure that the minimization does not produce the trivial partition with the entire set S as the only block in the partition.

3.1 Finding *Dense set* using $-w_e(\tilde{\mathrm{E}}(U)) - k$

The z.s.s. corresponding to $-w_e(\tilde{\mathrm{E}}(U)) - k$ is given by $\hat{f}(U) = k * w_v(U) - w_e(\tilde{\mathrm{E}}(U)) - k$. This z.s.s. is the *linearized* version of the measure of density of the subset U given by the ratio of the weight of edges with both endpoints in U and the weight of the vertex subset U.

To find out the *dense set* $U \subseteq V$, which has *edge density* atleast k, we would like to minimize this z.s.s., which is equivalent to minimizing $\hat{f}(U) = k * w_v(U) - w_e(\tilde{\mathrm{E}}(U))$. This minimization is achieved by finding max-flow (see [2] for details of maxfow-mincut algorithm) in network-flow formulation of the above problem, which is presented in the following sub-section.

3.2 Network-flow formulation for $\min_{U \subseteq V}(k * w_v(U) - w_e(\tilde{\mathrm{E}}(U)))$

Consider the hypergraph $H(V, E_h)$, as shown in Fig. 1. We represent this hypergrpah as a bipartite graph $G(V_1, V_2, E)$, where

- $V_1 \equiv$ Set of vertices, which has one vertex corresponding to each vertex of hypergraph H. The weight of such a vertex $u_1 \in V_1$ is equal to the weight of corresponding vertex $u \in V$.
- $V_2 \equiv$ Set of vertices, which has one vertex corresponding to each hyperedge of hypergraph H. The weight of such a vertex $u_2 \in V_2$ is equal to the weight of corresponding hyperedge $e \in E_h$.
- $E \equiv$ Set of directed edges, which connect a vertex $x \in V_1$ to a vertex $y \in V_2$, iff in hypergraph H, the hyperedge corresponding to the node y, is incident on the vertex corresponding to the vertex x.

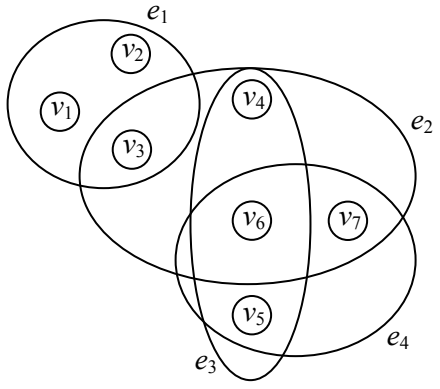

Fig. 1 – An example hypergraph consisting of hyperedges $e_1 = (v_1, v_2, v_3)$, $e_2 = (v_3, v_4, v_6, v_7)$, $e_3 = (v_4, v_5, v_6)$ and $e_4 = (v_5, v_6, v_7)$

As shown in Fig. 2, now we construct the *Flow Network* for this bipartite graph *G* as follows:

- Add a source vertex *s* and connect it to all nodes from the subset V_1 by a forward edge.
- Add a target vertex *t* and connect all nodes from subset V_2 to it by forward edges.
- Assign capacities of different types of edges as follows:
  - $\forall x \in V_1$, Cap $(s, x) = k * w_v(x)$
  - For each edge (x, y) such that $x \in V_1$ and $y \in V_2$, Cap $(x, y) = \infty$
  - $\forall y \in V_2$, Cap $(y, t) = w_e(y)$

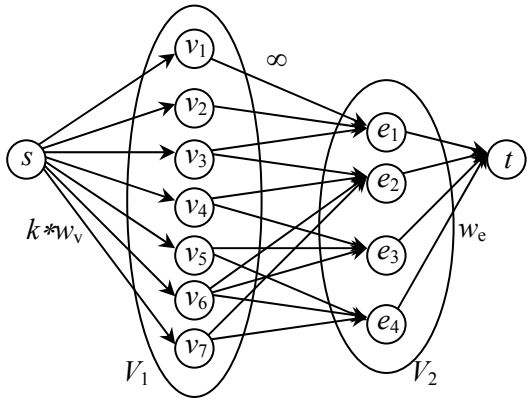

Fig. 2 – Flow Network corresponding to hypergraph shown in Fig. 1

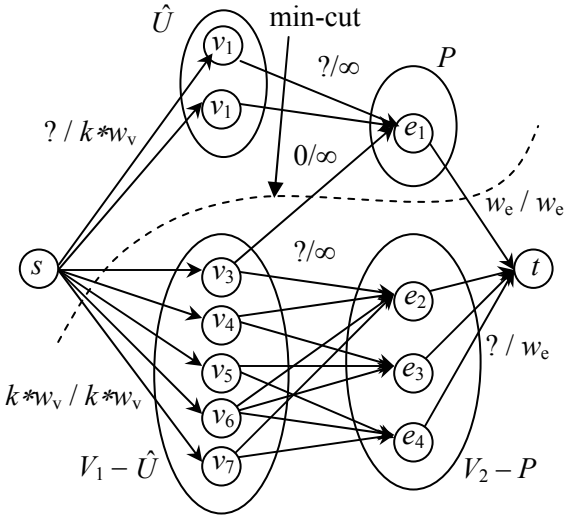

Fig. 3 – Typical scenario when max-flow is reached

A typical scenario, when the max-flow is reached (or equivalently min-cut is discovered) is shown in Fig. 3.

3.3 Justification of network-flow model

Now, we justify that using above network-flow formulation, we can indeed minimize $\left(k * w_v(U) - w_e\left(\tilde{\mathrm{E}}(U)\right)\right)$, $\forall U \subseteq V$.

Claim: There exists no forward edge from subset $\hat{U}$ to $(V_2 - P)$.

Proof: We know that when max-flow is reached, the capacity of the corresponding min-cut is reached, i.e. all forward edges from source-side to target-side of the min-cut are saturated and all backward edges carry *zero* flow.

But all edges, from subset V_1 to V_2 have *infinite* capacities, none of which can be saturated.

Claim: There exists no vertex in *P*, which is not connected to any vertex in $\hat{U}$.

Proof: As all forward edges from source-side to target-side of the min-cut are saturated, all edges from the subset *P* to target node *t* are saturated. And all such edges have $w_e(y), y \in P$ (*non-zero*) capacities.

Now let ∃ a vertex $p \in P$, which is not connected to any vertex from subset $\hat{U}$.

We know that the edge (p, t) carries $w_e(p)$ flow. But as *p* is not connected to any of the vertices from subset $\hat{U}$, the edge (p, t) has to receive this flow from some edge from the subset $(V_1 - \hat{U})$ to *p*. But all such edges are backward edges as far as min-cut is concerned, which carry *zero* flow. Thus, we get a contradiction!!

From above two claims, we can safely conclude that $P \cong \Gamma\left(\hat{U}\right)$, where $\Gamma(X)$ is defined as the set of all edges, which have atleast one end-point inside *X*.

Hence, can say that

$$Maxflow = w_e\left(\Gamma\left(\hat{U}\right)\right) + k * w_v\left(V_1 - \hat{U}\right)$$
$$= w_e(V_2) - w_e\left(\tilde{\mathrm{E}}\left(V_1 - \hat{U}\right)\right) + k * w_v\left(V_1 - \hat{U}\right)$$

$\Rightarrow Maxflow - w_e(V_2)$
$= k * w_v(V_1 - \hat{U}) - w_e(\tilde{E}(V_1 - \hat{U}))$

Now, as $w_e(V_2)$ is constant, $(Maxflow - w_e(V_2))$ minimizes $(k * w_v(U) - w_e(\tilde{E}(U)))$, $\forall U \subseteq V$.

4 Bipartition with cluster identification

Our heuristic approach to obtain a bipartition, which respects natural clusters, works as follows:

- First, it identifies as many natural *dense clusters* as possible, in the given circuit based on the ideas described in the previous section.
- Then, it provides hints to the FM algorithm to keep these clusters intact during the process of partitioning.

4.1 Algorithm to find out *dense cluster*

Now, we present the outline of the heuristic approach, which obtains the bipartition of the given VLSI netlist respecting the natural clusters inside.

Algorithm *Dense_FM_Hyp*

1. Let $H(V, E_h)$ be the given hypergraph, which is to be partitioned into two blocks. Now, we try to identify the *dense clusters* as follows:
   a. From *H*, we generate sub-hypergraphs of size of about 25 vertices based on the idea of proximity, i.e. once a vertex is chosen; its neighbor having maximum connectivity with the sub-graph constructed so far is added to it.
   b. For each sub-hypergraph generated, we try to identify the *dense cluster* inside it based on the ideas described in the previous section, with *k* equal to the density of the hypergraph itself.
2. For each such *dense cluster*, we insert one artificial hyperedge of heavy weight into the original hypergraph *H*, which connects all the vertices from that cluster. This discourages FM from breaking that cluster.
3. Use FM on this modified hypergraph to obtain its bipartition.

End Algorithm *Dense_FM_Hyp*

We have also devised the algorithm ***Dense_FM_Gr***, which converts the hypergraph $H(V, E_h)$ into its corresponding graph model *G*' (*V*', *E*') using clique model and then proceeds similar to algorithm ***Dense_FM_Hyp***.

We convert $H(V, E_h)$ into *G*' (*V*', *E*') as follows:

1. For each vertex $u \in V$ in *H*, add a vertex *u*' to *V*' in *G*. The weight *w*(*u*') is assigned same value as that of *w*(*u*).
2. Replace each hyperedge by its equivalent *clique* representation.

4.2 Analysis of the algorithm

As described in the outline of the algorithm ***Dense_FM_Hyp***, the sub-hypergraph consists of 25 vertices. Hence, the max-flow subroutine to identify *dense cluster* inside it runs very fast. We have found that the total time taken for finding all *dense clusters* is less than one run of UCLA_FM_CLIP (an implementation of FM with CLIP [4] available at http://vlsicad.eecs.umich.edu/BK/PDtools). And typically, we need 100 runs of UCLA_FM_CLIP to get best results, whereas this pre-processing of identifying *dense clusters* is done only once. So, it adds only 0.5%-1.0% to the time taken by 100 runs of UCLA_FM_CLIP.

5 Experimental Results

We have tested our approach on ISPD98 IBM Circuit Benchmark Suite (available at http://vlsicad.cs.ucla.edu/cheese). It is a collection of large circuits with number of cells ranging from 12K to 210K.

The results of experiments are presented in Table 1. The first three columns describe the characteristics of the circuits. The fourth column gives the best net-cuts obtained over 100 runs of UCLA_FM_CLIP (an implementation of FM with CLIP [4] available at http://vlsicad.eecs.umich.edu/BK/PDtools). The fifth and seventh columns give the best net-cuts obtained over 100 runs of our heuristic approaches, viz. ***Dense_FM_Hyp*** and ***Dense_FM_Gr***. The sixth and eighth columns give their corresponding improvements over UCLA_FM_CLIP.

As it is evident from Table 1, our heuristic approaches ***Dense_FM_Hyp*** and ***Dense_FM_Gr*** produce better results in atleast 50% of the test cases.

Circuit Name	Number of cells	Number of nets	UCLA_FM_ CLIP	Dense_FM _Hyp	improve-ment (%)	Dense_ FM_Gr	improve-ment (%)
ibm01	12752	14111	278	**251**	9.71	253	9.0
ibm02	19601	19584	**292**	301	-3.08	295	-1.03
ibm03	23136	27401	961	984	-2.39	**774**	19.46
ibm04	27507	31970	519	532	-2.50	**517**	0.39
ibm05	29347	28446	**1918**	1956	-1.98	1963	-2.35
ibm06	32498	34826	**622**	630	-1.29	669	-7.56
ibm07	45926	48117	**803**	809	-0.75	825	-2.74
ibm08	51309	50513	1356	1398	-3.10	**1312**	3.25
ibm09	53395	60902	**558**	560	-0.36	567	-1.61
ibm10	69429	75196	1104	**1041**	5.71	1100	0.36
ibm11	70558	81454	1023	**891**	12.90	947	7.43
ibm12	71076	77240	2785	**2330**	16.34	2334	16.19
ibm13	84199	99666	1280	**1031**	19.45	1179	7.89
ibm14	147605	152772	2277	**2242**	1.54	2260	0.75
ibm15	161570	186608	**2854**	3333	-16.78	3275	-14.75
ibm16	183184	190048	**2470**	2576	-4.29	2782	-12.63
ibm17	185495	189581	3651	**3141**	13.97	3529	3.34
ibm18	210613	201920	**2576**	2750	-6.75	2785	-8.11

Table 1 – Experimental Results

6 Conclusion

We have designed and implemented a new heuristic to obtain a bipartition of netlist, which respects natural *dense clusters*. Over 100 runs, this approach also produced bipartitions with better net-cut in atleast 50% of the ISPD98 benchmark circuits. Hence, we propose that this approach can be used to obtain a bipartition, which respects natural clusters and also has better net-cut.

References:

[1] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions", *Proc. of Design Automation Conference*, 1982, pp. 175-181.

[2] H. Narayanan, "Submodular Functions and Electrical Networks", *Annals of Discrete Mathematics, North Holland*, 1997.

[3] D. J.–H. Huang and A. B. Kahng, "When clusters meet partitions: new density-based methods for circuit decomposition", *Proc. of European conference on Design and Test*, 1995, pp. 60-64.

[4] Shantanu Dutt and Wenyong Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques", *Proc. of IEEE/ACM Int'l Conference on CAD*, 1996, pp. 194-200.