[LiLC95] Li, J., J. Lillis, C.K. Cheng, "Linear Decomposition Algorithm for VLSI Design Applications", *Proc. ACM/IEEE Design Automation Conf.*, pp. 223-228, 1995

[RiDJ94] Riess, B. M., K. Doll, F. M. Johannes, "Partitioning Very Large Circuits Using Analytical Placement Techniques", *Proc. ACM/IEEE 31st Design Automation Conf.*, June 1994

[Sa89] Sanchis, L., "Multiple-Way Network Partitioning", *IEEE Trans. on Computers*, pp. 62-81, 1989

[SeLe87] C. Sechen, K. W. Lee, "An Improved Simulated Annealing Algorithm for Row-Based Placement", *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 478-481, Nov. 1987

[ShNK95] Shibuya, T., I. Nitta, K. Kawamura, "SMINCUT: VLSI Placement Tool Using Min-Cut", *Fujitsu Scientific & Technical Journal*, pp. 197-207, 1995

[SrCK91] A. Srinivasan, K. Chaudhary, E. S. Kuh, "RITUAL: Performance Driven Placement Algorithm for Small Cell ICs", *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 48-51, 1991

[SuS93] Sun, W.-J., C. Sechen, "Efficient and Effective Placement for Very Large Circuits", *Proc. of ICCAD-93*, pp. 170-177, 1993

[SwSe95] Willian Swartz, Carl Sechen, "Timing Driven Placement for Large Standard Cell Circuits" *Proc. ACM/IEEE Design Automation Conference*, pp. 211-215, 1995

[TsKH88] Ren-Song Tsay, Ernest S. Kuh, and Chi-Ping Hsu, "PROUD: A Fast Sea-Of-Gates Placement Algorithm", *ACM/IEEE Design Automation Conference*, pp 318-323, 1988

[YaWo94] H. Yang and D. F. Wong, "Efficient Network Flow Based Min-Cut Balanced Partitioning", *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 50-55, 1994

[YeCL91] C. W. Yeh, C. K. Cheng, T. T. Lin, "A General Purpose Multiple-Way Partitioning Algorithm", *Proc. 28th ACM/IEEE Design Automation Conf.*, 1991

[YeCL92] C. W. Yeh, C. K. Cheng, T. T. Lin, "A Probabilistic Multicommodity-Flow Solution to Circuit Clustering Problems", *Int'l Conf. on Computer-Aided Design*, pp. 428-431, 1992

[ZiSC96] J.Y. Zien, M.D.F. Schlag, and P.K. Chan, "Multi-level Spectral Hypergraph Partitioning with Arbitrary Vertex Sizes", *Proc. ICCAD*, pp. 201-204, 1996

[CoLS96] Cong, J., W. Labio, and N. Shivakumar, "Multi-Way VLSI Circuit Partitioning Based on Dual Net Representation", *IEEE Trans. on Computer-Aided Design*, pp. 396-409, Apr. 1996

[CoSm93] Cong, J., M. Smith, "A Bottom-up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Designs", *ACM/IEEE Design Automation Conf.*, pp. 755-760, June 1993

[CoXu95] Cong, J., D. Xu, "Exploiting Signal Flow and Logic Dependency in Standard Cell Placement", *Proc. Asia and South Pacific Design Automation Conference* , pp. 399-404, 1995

[DoHo73] W.E. Donath, A.J. Hoffman, "Lower Bounds for the Partitioning of Graphs", *IBM J. Res. Dev.* pp. 420-425, 1973

[DuDe96] Dutt, S. and W. Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques", *Int'l Conf. on Computer-Aided Design*, pp. 194-200, 1996

[FiMa82] Fiduccia, C., R. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions", *ACM/IEEE Design Automation Conf.*, pp. 175-181, 1982

[GrSu84] Jonathan Greene, Kenneth Supowit, "Simulated Annealing without Rejected Moves", *Proc. Int'l Conf. on Computer-Aided Design*, pp. 658-663, 1984

[HaKa92] L. Hagen, A. B. Kahng, "A New Approach to Effective Circuit Clustering", *Int'l Conf. on Computer-Aided Design*, pp. 422-427, 1992

[HaHK95] L. Hagen, D. Huang, A.B. Kahng, "On Implementation Choices for Iterative Improvement Partitioning Algorithms", *IEEE Trans. on Comp. Aided Des.* 1995 (submitted)

[HuMo85] Hu, T., K. Moerder, "Multiterminal Flows in a Hypergraph", *VLSI Circuit Layout Theory and Design*, pp. 87-93, 1985

[HwGa92] J. Hwang, A. E. Gamal, "Optimal Replication for Min-Cut Partitioning", *Int'l Conf. on Computer-Aided Design*, pp. 432-435, 1992

[KeLi70] Kernighan, B., S. Lin, "An Efficient Heuristic Procedure for Partitioning of Electrical Circuits", *Bell System Technical Journal*, Feb. 1970

[KiGV83] Kirkpatrick, S., C. D. Gelatt, M. P. Vecchi, Jr. "Optimization by Simulated Annealing", *Science*, pp. 671-680, 1983

[KlSJ91] J. M. Kleinhans, G. Sigl, F. M. Johannes, K. J. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization", *IEEE Trans. on Computer-Aided Design*, pp. 356-365, 1991

[Kr84] Krishnamurthy, B., "An Improved Min-Cut Algorithm for Partitioning VLSI Networks", *IEEE Trans. on Computers*, pp. 438-446, 1984

[KrNe91] C. Kring, A. R. Newton, "A Cell-Replicating Approach to Mincu-Based Circuit Partitioning", *IEEE Int'l Conf. on Computer-Aided Design* pp. 2-5, 1991

demonstrate the superiority of our approach over the best available iterative partitioning algorithm CLIP-PROP_f [DuDe96] with an average improvement of 7.1% in terms of minimum cutsize.

Acknowledgement

This work was partially supported by DARPA/ITO under Contract J-FBI-93-112, NSF Young Investigator Award MIP-9357582, and Fujitsu Laboratories at America under the 1995 and 1996 California MICRO Programs.

References

[AlHK96] Charles J. Alpert, Lars W. Hagen, Andrew B. Kahng, "A Hybrid Multilevel/Genetic Approach For Circuit Partitioning", *Proc. ACM/SIGDA Physical Design Workshop*, pp. 100-105, 1996

[AlKa93] C. J. Alpert, A. B. Kahng, "Geometric Embeddings for Faster (and Better) Multi-Way Netlist Partitioning", *Proc. ACM/IEEE Design Automation Conf.* pp. 743-748, 1993

[AlKa95] C.J. Alpert, A.B. Kahng, "Recent directions in netlist partitioning: a survey", *Integration, the VLSI journal*, pp, 1-81, 1995

[AlYa95] C.J. Alpert, and S.-Z. Yao, "Spectral Partitioning: The More Eigenvectors, the Better", *Proc. ACM/IEEE Design Automation Conference*, pp. 195-200, 1995

[Ba82] E. R. Barnes, "An Algorithm for Partitioning the Nodes of a Graph", *SIAM J. Alg. Disc. Math.*, vol. 3, pp 541-550, 1982

[BeSa93] Mark Beardslee, Alberto Sangiovanni-Vincentelli, "Heuristic Methods for Communication-Based Logic Partitioning", *4th ACM/SIGDA Physical Design Workshop*, pp. 199-210, 1993

[Bo87] R.B. Boppana, "Eigenvalues and Graph Bisection: An Average-Case Analysis", *IEEE Symp. on Foundations of Computer Science*, pp 280-285, 1987

[CoDi93] Cong, J., and Y. Ding, "On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping", *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 213-218, June 1993

[CoHK91] Cong, J., L. Hagen, and A. B. Kahng, "Random Walks for Circuit Clustering", *IEEE 4th Int'l ASIC Conf.*, pp. 14-21, Sept. 1991

[CoHK92] Cong, J., L. Hagen, and A. B. Kahng, "Net Partitions Yield Better Module Partitions", *IEEE 29th Design Automation Conf.*, pp. 47-52, June 1992

[CoLB94] Cong, J., Z. Li, and R. Bagrodia, "Acyclic Multi-Way Partitioning of Boolean Networks", *Proc. ACM/IEEE 31st Design Automation Conf.*, pp. 670-675, June 1994

[CoLR90] Cormen, T., Leiserson, C., Rivest, R., "Introduction to Algorithm", *McGraw-Hill Book Co.*, 1990

circuit		*FBB*	*Paraboli*	*PANZA*	*FM-LSRb*
name	*size*	[YaWo94]	[RiDJ94]	[LiLC95]	-
s1423	619	13	16	15	13
s9234	5866	70	74	40	43
s13207	8772	74	91	66	58
s15850	10470	67	91	44	53
s35932	18148	49	62	43	47
s38584	20995	47	55	47	48
s38417	23949	58	49	49	51
sioo	664	-	45	25	25
s1488	686	-	50	44	42
balu	801	-	41	27	27
struct	1952	-	40	33	33
biomed	6514	-	135	83	84
TOTAL		378	749	516	524
RATIO		1.21	1.43	0.98	1.00

Table 4: Comparison to Non-Iterative Improvement Partitioners

# cells	# nets	# IOs	# fixed cells	normalized cell size variation	Fujitsu cutsize	FM-LSR cutsize
35762	36995	332	427	1 to 27	856	581

Table 5: Comparison to Fujitsu's Partitioner

5 Conclusion

In this paper, we introduced the notion and importance of *loose* net and provided a *loose* net removal (LR) algorithm to remove *loose* nets from the cutset. The *stable* net removal algorithm [ShNK95] was then incorporated into algorithm LR to form an efficient and powerful iterative circuit partitioning algorithm LSR for more global level control on hard-to-remove nets. Furthermore, we developed an efficient partitioning algorithm, called FM-LSRb, that combines a signal flow based hierarchical clustering algorithm using Maximum Fanout Free Subgraph (MFFS) technique with algorithm LSR. In algorithm FM-LSRb, the cluster decomposition strategy was also integrated to decompose the clusters that straddled on the cutline for further reduction of cutsize. Our research revealed that prior hierarchical clustering can be effectively incorporated into partitioning algorithm to improve the solution quality, even though the cluster removal technique was applied in the partitioning algorithm.

Our MFFS hierarchical clustering algorithm generalizes the MFFC clustering method [CoDi93, CoXu95] from combinational circuits to general sequential circuits and allows multiple outputs for each cluster so as to make signal flow based hierarchical clustering and declustering possible.

Our FM-LSRb algorithm has achieved best minimum cutsize results among all the iterative improvement based partitioning algorithms reported in the recent literature. The experiments on MCNC benchmarks

circuit		*LIFO-LA4*	*K-DualFM*	*GMetis*	*LA3-CDIP*	*CLIP-PROP$_f$*	*FM-LSRb*
name	*size*	[HaHK95]	[CoLS96]	[AlHK96]	[DuDe96]	[DuDe96]	-
balu	801	-	27	27	27	27	27
primary1	833	74	52	47	52	51	45
struct	1952	61	38	33	36	33	33
primary2	3014	259	-	142	152	152	120
s9234	5866	-	77	43	44	42	43
biomed	6514	165	107	102	83	84	84
s13207	8772	-	110	74	70	71	58
s15850	10470	-	93	53	67	56	53
s35932	18148	-	-	57	73	42	47
s38584	20995	-	-	53	47	51	48
avq.small	21918	658	-	144	148	144	143
s38417	23949	-	-	69	79	65	51
avq.large	25178	700	-	145	145	143	141
TOTAL		1917	504	989	1023	961	893
RATIO		3.39	1.47	1.11	1.15	1.08	1.00
TIME		-	-	2164.0	581.7	561.1	640.4

Table 3: Comparison to Iterative Improvement Partitioners

bucket, lookahead and cluster detection based), and CLIP-PROP$_f$ (CLIP, PROP, followed by PROP post-refinement). The experiment shows that FM-LSRb outperforms LIFO-LA4, K-DualFM, GMetis, LA3-CDIP, and CLIP-PROP$_f$ by 70.5%, 31.9%, 9.7%, 12.7%, and 7.1%, respectively. In particular, we obtained up to 21.5% (second largest example *s38417*) and average of 7.1% improvement over the best existing iterative improvement partitioner CLIP-PROP$_f$. This means that we have achieved the best cutsize result among all the iterative improvement based partitioning algorithms published in the literature. The $O(\lg n)$ runtime overhead due to the heap based bucket structure is negligible.

Table 4 shows the comparison of our FM-LSRb to non-iterative improvement partitioning algorithms FBB (network flow based), Paraboli (analytical placement based) and PANZA (network flow and spectral based). PANZA reports the best cutsize result among all categories of partitioning algorithms. However, the runtime report is omitted, and it also assumes uniform cell size instead of real size. Our FM-LSRb produces cutsize result as competitive as the state-of-the-art PANZA. In addition, FM-LSRb has the following advantages over other non-iterative improvement partitioning algorithms;

- Area balance constraint as well as various cell sizes can be easily accommodated.
- Preassignment of certain cells to specific blocks can be easily implemented by locking them into the blocks during the entire partitioning process.
- Extension to multiway partitioning is possible straightforward.
- One can easily control the runtime vs cutsize trade-off.

We have applied our FM-LSR algorithm to an industrial benchmark from Fujitsu Laboratories Ltd. of 35762 cells with variable cell size and fixed cell constraint. Our FM-LSR outperforms Fujitsu's current partitioning tool by 32.1% in terms of minimum cutsize. The detailed result is shown in Table 5.

circuit		basic	LR based			LSR based		
name	*size*	FM_{100}	*LR*	*LRn*	*LRb*	*LSR*	*LSRn*	*LSRb*
s1423	619	15	13	13	13	13	13	13
sioo	664	27	25	25	25	25	25	25
s1488	686	46	43	43	42	45	44	42
balu	801	27	27	27	27	27	27	27
primary1	833	50	45	47	46	45	45	45
struct	1952	51	33	33	33	33	33	33
primary2	3014	173	120	123	122	121	121	120
s9234	5866	57	46	45	44	43	43	43
biomed	6514	164	84	84	84	84	84	84
s13207	8772	101	76	74	72	67	62	58
s15850	10470	94	61	60	57	66	57	53
s35932	18148	61	47	47	47	45	48	47
s38584	20995	64	49	49	49	49	49	48
avq.small	21918	231	156	150	140	141	148	143
s38417	23949	343	52	51	51	71	61	51
avq.large	25178	241	171	159	144	153	147	141
TOTAL		1745	1048	1030	996	1028	1007	973
RATIO		1.79	1.08	1.06	1.02	1.06	1.03	1.00
TIME		967.2	575.5	432.0	702.2	512.9	349.5	640.4

Table 2: Cutsize Reduction from Various Enhancements

result reveals the following;

- Prior-clustering as well as declustering can be effectively incorporated into LR/LSR to improve the partitioning solution quality even further. This is evident from the cutsize reduction trend along L(S)R $\rightarrow$ L(S)Rn $\rightarrow$ L(S)Rb.
- The idea of combining the static *stable* net removal with dynamic *loose* net removal (LSR) is very effective in removing nets in the cutset. This is evident from the cutsize comparison between LR based algorithms (LR, LRn, LRb) and LSR based algorithms (LSR, LSRn, LSRb).
- The hierarchical MFFS clustering reduces the complexity of the given netlist and thus the partitioning time. This is evident from the runtime comparison between L(S)R and L(S)Rn.
- LSR based algorithms are always faster than LR based algorithms since SNT promotes higher rate of convergence due to its minor perturbation of the current partition compared to an entirely new random initial partition.

4.2 Comparison to Other Partitioning Algorithms

Table 3 shows the comparison of our heap based FM-LSRb to state-of-the-art iterative improvement partitioning algorithms LIFO-LA4 (LIFO bucket and modified lookahead formulation based), K-DualFM (dual net representation based), GMetis (genetic multi-level graph partitioning based), LA3-CDIP (LIFO

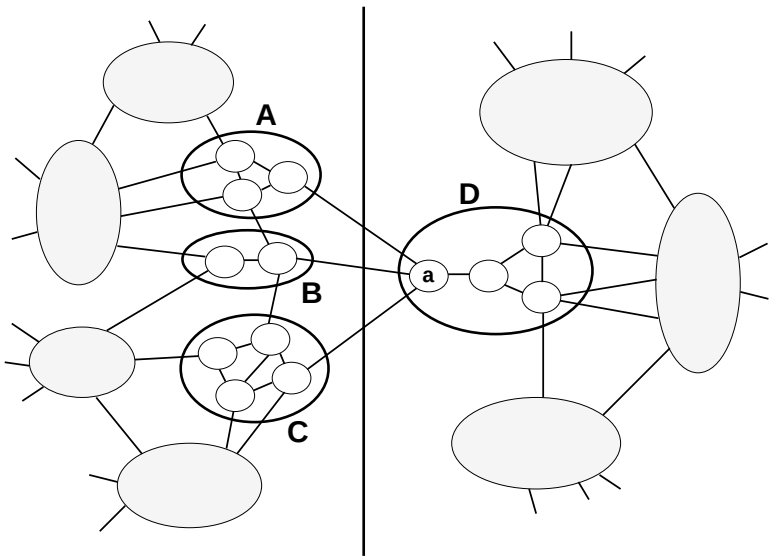


Figure 4: Benefit of Boundary Declustering

LSR, the *hierarchical boundary declustering* is developed. At the end of LSR run, only the clusters that are connected to the nets in the cutset, say A through D in Figure 4, are decomposed into the next level of hierarchy for additional attempt of LSR run in order to give finer-grained attention to nets that are cut. This cooperation of hierarchical boundary declustering and LSR based partitioning continues until no more clusters can be decomposed. The balance of the hierarchy tree as well as the size limit of clusters for each level of hierarchy is an important issue for the successful adoption of declustering into partitioning.

4 Experimental Result

We have implemented heap based LR, LSR, hierarchical MFFS clustering and declustering on Unix based SUN SPARC workstation. Our algorithms were tested with the same set of benchmark circuits used in [DuDe96] (some of them are omitted since they do not have the signal directions, which are required by our MFFS clustering algorithm). The bipartitioning result is based on the minimum cutsize of 20 runs under 45-55% area balance criterion and real module sizes. The average runtime is measured in seconds using SPARC station 5 Model 85.

4.1 Cutsize Reduction Trend

The experimental result summerized in Table 2 shows the cutsize reduction trend starting from the basic FM to our most enhanced FM-LSRb. FM_{100} refers to the 100 runs of basic FM with conventional bucket structure, while all the other algorithms are based on 20 runs. L(S)Rn refers to the L(S)R tested on hierarchical MFFS clustered netlist followed by 'n'o declustering. Likewise, L(S)Rb refers to the L(S)R tested on hierarchical MFFS clustered netlist followed by hierarchical 'b'oundary declustering. The experimental

$$N15 = \{N9\}$$

The graph representation of the clustered netlist is shown in Figure 3(d).

To do higher level MFFS clustering for the clustered circuit in Figure 3(d), we go through all nodes in Figure 3(d) to build higher level MFFS clusters. Let us take $N14$ in Figure 3(d) as an example. Node $N14$ fanouts to node $N11$ and $N12$ (Note that the edge direction is reversed in Figure 3(d)). So, we choose them as the root nodes. We cut the two input edges, e.g. edge ($PO1$, $N11$) and edge ($PO2$, $N12$), of node $N11$ and $N12$ and traverse the graph of Figure 3(d) from $PO3$. Node $N11$, $N12$ and $N14$ cannot be reached. So, we obtain a multiple output MFFS cluster $\{N11, N12, N14\}$. By looking at the other nodes in Figure 3(d), we can obtain following clusters:

$$\begin{aligned} m_1 &= \{N11, N12, N14\} \\ m_2 &= \{N12, N13, N15\} \\ m_3 &= \{N11\} \\ m_4 &= \{N12\} \\ m_5 &= \{N13\} \end{aligned}$$

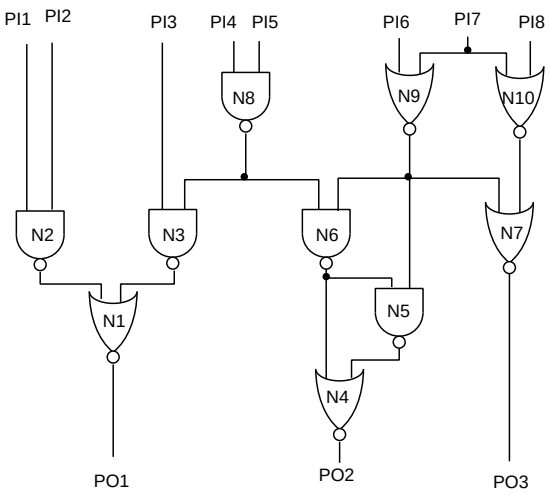
Since m_1, m_2 share node $N12$, there is an edge between these two nodes in the sharing graph $G_{share}(SN, SE)$, as shown in Figure 3(e). Similarly, we can set all the other edges in the sharing graph. The cost calculated by the cluster density metric is also shown in Figure 3(e). Finally, m_2 and m_3 will be chosen to minimize the total sum of the cost without conflict of node sharing. The remaining node $N14$ in Figure 3(e) will then be transfer to the higher level clustering hierarchy directly without making any change. So, we finally obtain following higher level clusters;

$$\begin{aligned} m_2 &= \{N12, N13, N15\} \\ m_3 &= \{N11\} \\ m_6 &= \{N14\} \end{aligned}$$

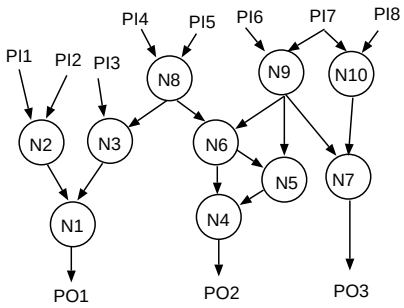
3.5 Hierarchical Boundary Declustering

As mentioned earlier, LR is an effective local optimizer that focuses on smaller nets, whereas more global level of optimization can be accomplished through the handling of larger nets with LSR. However, our research revealed that the nets in the cutset still requires more extensive attention compared to nets that are not cut, because they have higher probability of staying stable even in the presence of stable net detection and removal. As a consequence, we realized that the decomposition of clusters that are related to the nets in the cutset can dramatically improve the solution quality due to the additional freedom of cell movement. If we decompose clusters A through D in Figure 4, subcluster a can be moved to reduce the cutsize by $3 - 1 = 2$.

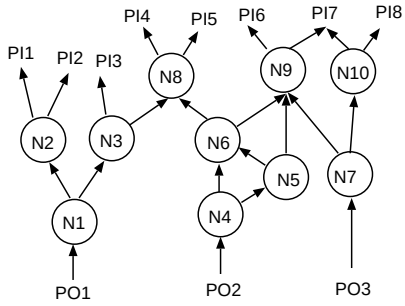
The multiple output MFFS clustering algorithm can form a hierarchical clustering tree after a recursive clustering process, which permits hierarchical declustering where each cluster can be decomposed into subclusters. Among many possible strategies of decomposing the cluster hierarchy that can work well with



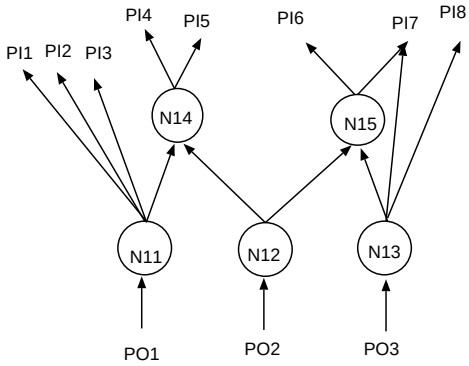
(a) Circuit



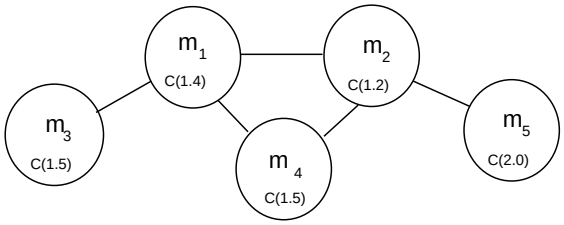
(b) G(N,E)



(c) G'(N,E')



(d) G''(N',E'')



(e) G_share(SN, SE)

Figure 3: Hierarchical Clustering Example

(d) We choose a set of clusters SC, $SC \subseteq SN$, such that the total sum of the cost for the clusters in SC is minimized and any two clusters in SC are not connected by any edges in graph G_{share}. Since this problem is equivalent to the maximum independent set problem which is NP-complete problem, we use a simple heuristic to solve it as follows: First, the clusters in SN are sorted in increasing order of their costs. Then, the cluster with minimum cost is selected from SN, and all the neighbors of this cluster in graph G_{share} are removed from SN. This process is repeated until there is no candidate cluster in SN.

(e) Finally, if some nodes in N_c are not included in the higher level MFFS clusters in SC ($SC \subseteq SN$) (these nodes are called the remaining nodes), we transfer them to the higher level clusters directly.

The MFFS clustering algorithm can be applied multiple times to establish a number of desired hierarchy levels for circuit clustering. In the MFFS clustering algorithm, the time complexity for doing bottom level clustering is $O(|N| \times (|N| + |E|))$ in worst case, where $|N|$ is the number of nodes and $|E|$ is the number of edges in the directed graph, $G(N, E)$. The time complexity for doing higher level MFFS clustering is $O(|N_c| \times (|N_c| + |E_c|))$, where N_c is a set of nodes and E_c is the number of edges in graph $G_c(N_c, E_c)$ ($G_c(N_c, E_c)$ is the graph representation of the clustered netlist NL_c).

Theorem 2 *The time complexity of the MFFS clustering algorithm is* $O(|N| \times (|N| + |E|))$ *in the worst case.*

3.4 Example

We use an example to illustrate our hierarchical MFFS clustering algorithm in Figure 3. Figure 3(b) shows the graph representation $G(N, E)$ ($N = \{N1, N2, ..., N10\}$) of the circuit in Figure 3(a). Figure 3(c) shows graph $G'(N, E')$ which reverses the direction of edges of graph $G(N, E)$.

At the beginning, we do single output MFFS clustering to obtain the bottom level MFFS clusters. In this example, the root node set is $roots = \{N1, N4, N7\}$. We select a root node from $roots$ each time and then construct a single output MFFS cluster until there is no nodes left in $roots$. For example, in Figure 3(c), suppose $N1$ is selected as a root node, the edge from $PO1$ to $N1$ will be cut. Then the nodes, $N1$, $N2$ and $N3$ will be included in a MFFS, because these three nodes cannot be reached when graph G' is traversed and labeled from primary output $PO2$ and $PO3$. After that, another root node is selected, and we use the same method to find its single output MFFS cluster. We keep doing this until all nodes in $roots$ are exhausted. Then, we obtain single output MFFS clusters as follows for this example:

$$N11 = \{N1, N2, N3\}$$
$$N12 = \{N4, N5, N6\}$$
$$N13 = \{N7, N10\}$$

After that, we take out all nodes in $N11$, $N12$, and $N13$ from the netlist and obtain the shrunk circuit. By using the same approach, we can obtain the single output MFFS clusters of the shrunk circuit as follows:

$$N14 = \{N8\}$$

Theorem 1 *The time complexity of the MFFS construction algorithm is $O(|N| + |E|)$, where $|N|$ is the number of nodes and $|E|$ is the number of edges in the directed graph, G(N, E).*

3.3 MFFS Clustering Algorithm

The MFFS construction algorithm tells us how to obtain one MFFS cluster. If we want to cluster the entire netlist, we need to apply MFFS construction algorithm repeatedly. In this case, the selection of the root node set is very important. Our MFFS clustering algorithm works as follows:

1. We do single output MFFS clustering first to obtain the bottom level clusters. For a given netlist NL, the roots of its single output MFFS clusters are the fanin nodes of PO's at the beginning. Assume those root nodes are put in set *roots*. When we take out a node $root_i$ ($root_i \in roots$) from *roots* and apply the MFFS construction algorithm. A single output MFFS cluster for root node $root_i$ can be obtained. This process can be done repeatedly until *roots* is empty. Then, we take out all the currently obtained MFFS clusters out of the netlist NL, and obtained a shrunk netlist NL_s. We repeat the same procedure for the new netlist NL_s recursively until all cells in netlist NL are grouped into MFFS clusters. Finally, we obtain the clustered netlist NL_c after single output MFFS clustering, which is the basis of our hierarchical clustering procedure.

2. For the higher level MFFS clustering, we do clustering on top of the clustered netlist NL_c. Suppose the corresponding graph representation of this clustered netlist is $G_c(N_c, E_c)$, where N_c is a set of nodes which correspond to clusters in the clustered netlist NL_c, and E_c is a set of directed edges. A directed edge (n_i, n_j) exists if node n_i is a fanin of node n_j. Our higher level MFFS clustering is implemented as follows:

   (a) For each node n_i ($n_i \in N_c$), all the nodes that are fanouts of this node are considered as the root nodes. If the number of those nodes is larger than 2, we can use the MFFS construction algorithm to obtain a multiple output MFFS cluster; otherwise we can obtain a single output MFFS cluster. We keep doing this procedure until each node in N_c has a corresponding higher level MFFS cluster.

   (b) Note that, the resulting higher level MFFS clusters may not be realized at the same time, because it is possible that a node in N_c can be included in more than one higher level MFFS clusters. So, we define a sharing graph $G_{share}(SN, SE)$, where SN is a set of nodes ($m_i \in SN$) (m_i represents a higher level MFFS cluster which consists of a number of nodes in N_c), and SE is a set of edges where an edge (m_i, m_j), ($(m_i, m_j) \in SE$), exists if the corresponding clusters m_i and m_j share the same node in N_c.

   (c) To choose an appropriate set of higher level MFFS clusters from SN, we calculate the cost of each higher level cluster m_i ($m_i \in SN$) according to the connectivity information of the cluster. We use a cluster density metric as the cost in our current implementation. The definition of cluster density metric is given as $|N_{m_i}|/|E_{m_i}|$, where $|N_{m_i}|$ is the number of nodes included in a cluster m_i and $|E_{m_i}|$ is sum of the weights of edges in the cluster.

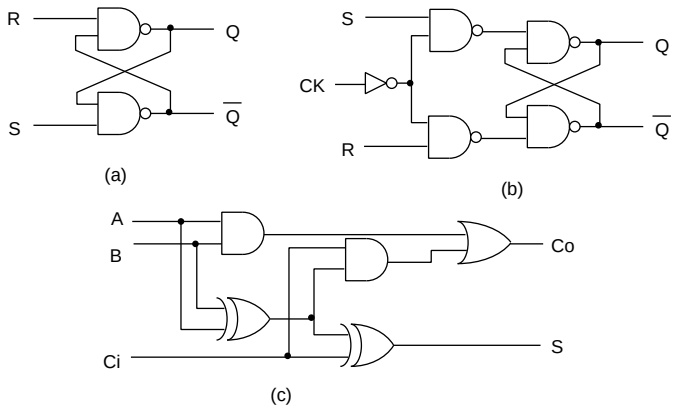


Figure 2: Multi-Output Circuit Group

Now, it is natural for us to extend FFC and MFFC to Fanout Free Subgraphs (FFS) and Maximum Fanout Free Subgraphs (MFFS) for the clustering of general sequential circuits.

Definition 3 (FFS & MFFS) *For a given node set S in a sequential circuit,*

$$FFS_S = \{u|\ \textit{every path from } u \textit{ to some PO passes through at least one node in } S \textit{ in the circuit}\}$$
$$MFFS_S = \{u|\ \textit{for all } FFS_S, u \in FFS_S\}$$

where S is called the root set of $MFFS_S$. If there are more than one nodes in set S, we call this $MFFS_S$ as a multiple output MFFS cluster; otherwise, we call it as a single output MFFS cluster.

This extension has two advantages. First, since it allows more than one outputs for each cluster, it is possible to group some tightly connected logic structures into clusters, such as the RS latch (Figure 2(a)), the clocked RS flip-flop (Figure 2(b)), or the full-adder (Figure 2(c)), constructed using simple gates in combinational or sequential circuits. Second, we shall show that the MFFS clustering can be computed as efficiently as MFFC clustering.

3.2 MFFS Construction Algorithm

For a given netlist NL and a root node set S, the MFFS cluster can be obtained by using the following simple and fast approach:

1. Convert the given netlist to a directed graph, $G(N, E)$, where N is a set of nodes which correspond to the cells in the netlist, and E is a set of directed edges. A directed edge (i, j) exists if node i is a fanin of node j.

2. Cut all the fanout edges of the root nodes in S; search all other nodes in graph $G(N, E)$ starting from the primary outputs (PO) of the netlist in the inverse direction of edges. The nodes in $G(N, E)$ that are not traversed belong to the $MFFS_S$.

This algorithm was named as the MFFS construction algorithm.

produce better cutsize result than starting from a single random initial partition with 20 times of SNT search. In addition, we observe that the idea of *stable* net removal can be combined perfectly with the idea of *loose* net removal discussed in the previous section. Loose net removal is a dynamic net removal method activated in the middle of a run whereas *stable* net removal regularly removes nets from the cutset after each run. LR is an effective local optimizer that focuses on smaller nets, whereas more global optimization can be accomplished via removing larger *stable* nets. The result of combining two ideas is the partitioning algorithm named Loose and Stable-net Removal (LSR).

3 MFFS Based Clustering Algorithm

Efficient clustering algorithms are essential to improve the quality of the partitioning results and to speed up partitioning algorithms by reducing the number of nodes to be partitioned. In this section, we present a signal flow and logic dependency based clustering algorithm MFFS, which generalizes existing MFFC decomposition method [CoDi93] from combinational circuits to general sequential circuits.

The MFFC based clustering algorithm has been reported to provide natural clustering solution for combinational circuits [CoDi93, CoLB94]. A modified MFFC algorithm for sequential circuits has been presented in [CoXu95], but the capability of this algorithm to detect directed cycles is limited. In general, the MFFC algorithm has the following restrictions; (i) it is mainly applied to combinational circuits; (ii) it allows only one output signal for each cluster, which may result in many small clusters. In the following section, we introduce an approach based on Maximum Fanout Free Subgraph (MFFS), which can overcome the two restrictions of MFFC clustering.

3.1 Definition of MFFS

We first rewrite the definition of FFC and MFFC, and then present the definitions of FFS and MFFS. The original definitions of FFC and MFFC are as follows [CoDi93]; Let $input(v)$ denote the set of nodes which are the fanins of node v, and $output(v)$ the set of nodes which are the fanouts of node v. For a node v in the circuit, a cone of v, denoted by C_v, is a subgraph of logic gates consisting of v and its predecessors such that any path connecting a node in C_v and v lies entirely in C_v. We call v the root of C_v.

Definition 1 (FFC & MFFC) *A fanout free cone at v, denoted by FFC_v, is a cone of v such that for any node $u \neq v$ in FFC_v, output(u) $\subseteq FFC_v$. The maximum fanout free cone of v, denoted by $MFFC_v$, is an FFC of v such that for any non-PI node w, if output(w) $\subseteq MFFC_v$, then $w \in MFFC_v$. We call v the root of FFC_v or $MFFC_v$.*

This definition implies that every path from u passes through v if u is included in FFC_v or $MFFC_v$. Therefore, it can be rewritten as:

Definition 2 (FFC & MFFC) *For a given node v in a combinational circuit,*

$$\begin{aligned} FFC_v &= \{u|\ \textit{every path from } u \textit{ to some PO passes through } v \textit{ in the circuit}\} \\ MFFC_v &= \{u|\ \textit{for all } FFC_v, u \in FFC_v\} \end{aligned}$$

circuit		minimum cutsize of 20 runs				
name	*size*	*FM*	*LA3*	*LA3-CLIP*	*LA3-CDIP*	*FM-LR*
s1423	619	17	16	16	15	13
sioo	664	31	25	25	25	25
s1488	686	48	42	42	41	43
balu	801	27	27	27	27	27
primary1	833	47	52	52	52	45
test04	1515	87	82	51	52	45
test03	1607	75	80	57	57	59
test02	1663	149	126	92	90	93
test06	1752	67	70	60	60	64
struct	1952	46	44	33	36	33
test05	2595	127	99	80	74	43
primary2	3014	212	149	142	152	120
s9234	5866	59	43	47	44	46
biomed	6514	83	90	84	83	84
s13207	8772	98	85	68	70	76
s15850	10470	109	87	73	67	61
s35932	18148	85	168	73	73	47
s38584	20995	100	85	55	47	49
avq.small	21918	347	608	146	148	156
s38417	23949	240	284	73	79	52
avq.large	25178	350	398	138	145	171
TOTAL		2404	2660	1434	1437	1352
RATIO		1.78	1.97	1.06	1.06	1.00
TIME		-	-	318.86	611.30	589.34

Table 1: Cluster Removal Strategies

After a run of FM type algorithm terminates, the initial and final cutsets are compared to detect *stable* nets. Then a *stable* net is randomly picked and all of its cells are moved into the smallest block, causing the *stable* net to be manually removed from the cutset. If a cell is moved once during this process, it cannot be moved to other blocks during a handling of another *stable* net. This process is repreated until a predetermined percentage of the *stable* net is handled or no more move is possible, then another run of FM is tried with the outcome of the *stable* net removal. A net is defined to be *true stable* if it remains cut during multiple runs of FM, and these nets are given higher priority during the *stable* net selection due to the fact that the earlier the net is chosen, the less possible it has cells that have already been processed. Usually the cutsize increases right after removing the *stable* net, but the degree of hill-climbing is not so costly because SNT promotes higher rate of convergence and shorter duration of runs due to the fact that changing the arrangement of cells of only the *stable* nets would result in a minor perturbation of the current configuration compared to an entirely new random initial configuration.

The original SNT algorithm [ShNK95] begins with only one random initial partition and evolves from it. However, our research shows that a better partitioning result can be obtained if SNT based search is performed on multiple random initial points under fewer iterations for each run. For example, starting from 2 random initial partitions and performing SNT search 10 times for each partition turned out to

```
single_pass_LR()
while (gain > 0)
  c = pick cell with maximum gain;
  move and lock cell c;
  for (each net n incident to cell c)
      update gain of cells in net n;
      if (n is loose net)
          for (each free cell f of net n not in anchor)
              f.gain = f.gain + incr(n);
endwhile
```

Figure 1: Loose-net Removal Algorithm

The *heap* data structure is a partially ordered array object that supports insertion and deletion of the element and update of values of the element with $O(\lg n)$ access time due to the corresponding partial re-ordering [CoLR90], where n is the total number of the elements. The conventional bucket structure for bipartitioning requires $O(p+n)$ amount of memory with $O(p)$ access time [FiMa82]. However, heap is independent of p since it requires $O(n)$ amount of memory with $O(\lg n)$ access time. Since the cell gain range is extended due to LR, heap is a reasonable choice for storing cell gains while maintaining the memory efficiency. The access time overhead is also reasonable. For example, our heap based FM running on SUN SPARC5 spent 115.4 sec for bipartitioning the 16 test circuits we used, whereas Dutt and Deng's conventional bucket based FM tested on the same machine spent 97.3 sec [DuDe96].

In addition, the LIFO bucket structure can easily be simulated with heap in the following way. We can maintain the *global move counter* that keeps track of how many cell moves have occurred so far. Whenever the gain of a cell is updated, the global move counter associated with the cell is also updated with the current value. When the comparison of two cell gains ends in a tie, we compare the move counter of the two cells to determine the one with the latest gain update.

The experimental result summerized in Table 1 reveals the effectiveness of LR as a cluster remover. The minimum cutsize of 20 runs for FM, LA3, LIFO-LA3-CLIP, and LIFO-LA3-CDIP reported in [DuDe96] are used for the comparison. As one can see, with the very simple strategy of LR, we achieved from 5.7% to 49.2% improvement over the previous iterative improvement partitioning algorithms. This convinces us that our strong form of cluster removal strategy focusing on the free cells of loose nets is shown to be very effective without any kind of tie-breaking or *locality* promoting enhancement. The authors of [DuDe96] provided a discussion about why LA3 fails on big circuits.

2.2.3 Loose and Stable-net Removal

A net is defined to be *stable* if it has remained cut throughout the entire run. Shibuya et al[ShNK95] observed that more than 80% of the nets in the final cutset are *stable*, and these nets trap FM type algorithm into local minima and limit the solution quality. From this observation, they proposed an hill-climbing method called *Stable Net Transition* (SNT) that enables FM type algorithm to discover multiple sub-optimal solutions within a practical time.

2.2.1 Loose-net Removal

A *free* net contains only free cells, and a *locked* net has locked cells in two or more blocks. Then, a net is defined to be a *loose* net if it has locked cells in exactly one block and at least one free cell in another block. *Anchor* block of a loose net n is defined as the block that contains the locked cells of n, and all the other blocks that have one or more free cells of n form *tail* blocks. Once a cell of a free net is moved and locked into a block, the net immediately becomes a *loose* net with the block as its anchor. But if some of the remaining cells of tail blocks are moved and locked at non-anchor blocks, the net will remain locked into the cutset thereafter during the entire pass. In other words, every net in the final cutset goes through the $free \rightarrow loose \rightarrow locked$ cycle, so our primary goal is to prevent as many *loose* nets from becoming locked as possible.

The basic idea of Loose-net Removal (LR) algorithm is to intentionally increase the cell gain of free cells of a *loose* net located outside the anchor block while they remain free. Those cells are then encouraged to move to the anchor block during the subsequent moves and thus the corresponding *loose* net is not locked into the cutset. For the free cells of a *loose* net located inside the anchor block, we have two choices; lock it inside the anchor block and do not move, or let the partitioning algorithm decide what to do with them. Since those cells can be involved in removing other *loose* nets, the latter strategy entails more freedom, and this argument is supported by our experiments.

After each cell move, LR considers all *loose* nets that are incident to the moved cell. The magnitude of cell gain increase due to a given *loose* net n is computed as follows;

$$incr(n) = \left\lfloor \frac{max_net_size}{size(n)} \cdot \frac{\sum_{c \in L_n} deg(c)}{\sum_{c \in F_n} deg(c)} \right\rfloor$$

where $size(n)$ denotes the number of cells of n, $deg(c)$ denotes the number of nets incident to cell c, and max_net_size denotes the size of the largest net in the given netlist. L_n denotes the set of all locked cells located inside the anchor block of n, and F_n denotes the set of all free cells located outside the anchor block of n. The above equation favors small nets, dense connectivity at the anchor, and sparse connectivity at the tail to encourage the algorithm to focus on relatively easy-to-remove *loose* nets. The LR algorithm based on the modified gain computation is described in Figure 1. Note that this algorithm can be extended to consider lookahead easily, but our experiment shows that lookahead in LR is not necessary.

2.2.2 Heap Based Bucket Structure

The conventional cell gain ranges from $-p$ to p, where p represents the maximum number of nets incident to a single cell. Therefore, the conventional bucket structure consists of an array of $2p+1$ entries to which cells with the same gain are linked. Since the entire cells are distributed into this short range, many cells will have the same gain values, requiring a better tie-breaking strategy such as the lookahead capability [Kr84]. LR causes the cell gain range to expand, eliminating the necessity of lookahead. However, the extended gain range makes the bucket structure memory inefficient, so we use the *heap based bucket structure* for storing cell gains.

is attempted using the result of the current pass as its initial partition. We keep activating passes until there is no further improvement in terms of cutsize. This process that consists of multiple passes is then called a *run*, and a new run can be attempted with another random initial partition. FM based algorithms can be implemented in linear time with the aid of special data structure called *bucket* that supports fast update and retrieval of cell gains.

2.1.2 Previous Cluster Removal Strategies

The *locality* of cell moves is established if all the *neighbors* of the currently moved cell are chosen to move subsequently. Then, it was Hagen et al [HaHK95] who first observed that this *locality* motivates the removal of closely connected cells that straddle the cutline. In order to promote this *locality*, they suggested the LIFO (Last In First Out) order of organizing the cells with the same gain. A *bucket* refers to a linked list of cells with the same gain. A LIFO bucket always puts the most recently moved cell at the head of the linked list. Therefore, the LIFO bucket organization ensures the cells with the most recently updated gain to be chosen first among others. However, this is a weak form of encouraging the neighboring cells to be chosen subsequently since the *locality* can easily be destroyed from the update of other cells with even higher gain values.

Dutt and Deng [DuDe96] recently proposed a more efficient way of promoting the *locality* in the choice of cells to move subsequently. They viewed the cell gain as the sum of *initial* and *updated* gain, where the former refers to the gain computed for each cell before any move, and the latter refers to the sum of dynamically updated gain values from the move of cells afterwards. Their strategy is based on the observation that the gain update is applied only to the cells of nets that are incident to the currently chosen cell. So focusing only on the updated part of gain for choosing the next cell encourages the neighboring cells to move subsequently. In order to focus on the updated gains, they initialized all the gain values to be zero after picking up the very first cell to move according to the initial gain. As a result, their algorithm CLIP/CDIP produced the best cutsize result among all the published iterative improvement type partitioning algorithms so far. However, their strategy is not free from frequent tie-breaking situation, which still requires the lookahead capability such as LA3 [Kr84] for better tie-breaking. Furthermore, they still rely on LIFO buckets to boost their ability to minimize the cutsize.

2.2 LSR Partitioning Algorithm

We present in this section even stronger form of cluster removal strategy by gradually increasing the gains of the neighbors connected to the currently chosen cell via *loose* nets until all of them are moved to one block. This strategy encourages the neighboring cells to be chosen subsequently and thus maintaining the *locality*. It also reduces the frequency of tie-breaking situation since the rank of cells in the bucket is determined according to the increased gain values. We shall first present the concept of *loose* nets and its relation to cluster removal strategy. Then we present a modified cell gain computation and Loose-net Removal (LR) algorithm along with its enhancement with the combination of *Stable Net Transition* [ShNK95].

by 17.2% and 30.0%, respectively.

The remainder of the paper is organized as follows. Section 2 presents the definitions, formulations, and algorithm on circuit partitioning. Section 3 presents signal flow based hierarchical clustering algorithm. Section 4 provides experimental results. Section 5 concludes the paper.

2 LSR Based Partitioning Algorithm

2.1 Circuit Partitioning Formulation

In this section, the formulation of iterative improvement type partitioning based on the cell gain concept is presented. Then recently published strategies to remove clusters that straddle the cutline are discussed.

2.1.1 Iterative Improvement Partitioning

Given a netlist NL to be partitioned into K partitions (also called *blocks* in the literature), we use $C = \{c_1, c_2, ..., c_p\}$ to denote the set of cells in NL, $N = \{n_1, n_2, ..., n_q\}$ to denote the set of nets in NL, and $P_1, P_2, ..., P_K$ to denote the K partitions, where p is the number of cells, and q is the number of nets in NL. The cells may have different areas. A net is said to be *cut* if it has at least one cell in more than two partitions, and all nets that are cut form a set called *cutset* whose cardinality is called *cutsize*.

An *optimal area-balanced K-way partitioning solution* of a given netlist NL satisfies the following conditions;

- Each cell is assigned to exactly one partition;
- The total area of the cells in each partition is within the following bounds;

$$(1-\alpha)\cdot\frac{A}{K} \leq A_i \leq (1+\alpha)\cdot\frac{A}{K}$$

  where A is the total area of all the cells in NL, A_i is the total area of all the cells in partition P_i, and α is a user-specified parameter controlling the allowable slack in the area constraint ($0 < \alpha < 1$);
- The cutsize is minimized.

An *iterative improvement* type of partitioning algorithm such as FM [FiMa82] uses the notion of *cell gain* to represent the reduction in the cutsize if the cell is moved to another block. Starting from a random initial *partition* which defines one-to-one correspondence between a cell and a block, the algorithm calculates the gain of entire cells and chooses the cell with maximum gain that satisfies the size constraint to be moved to another block. A cell is labeled *free* if it has not been moved so far and *locked* otherwise. *Neighbors* of a cell refer to all the free cells that connect to the given cell. After the currently chosen cell is moved and labeled locked, update of cell gains is applied to all free cells that are contained in the nets that incident to the moved cell. Then, the next maximum gain cell that satisfies the area constraint is chosen and moved. This process continues until all cells are moved. Then we backtrack the move sequence to retrieve the one that produces the minimum cutsize. This entire process is called a *pass*, and a new pass

nets in the cutset through our specially designed FM-LSRb algorithm. Our FM-LSRb algorithm combines hierarchical clustering based on Maximum Fanout Free Subgraph (MFFS) clustering with a Loose and Stable net Removal algorithm (LSR). In our iterative improvement partitioning algorithm, we focus on the status of nets, instead of cells as often emphasized in the previous approaches, in each *pass* (where, a *pass* is the entire moving process for all unlocked cells to become locked in an iterative improvement partitioning algorithm). Every nets in the cutset follow $free \rightarrow loose \rightarrow locked$ cycle, where a *free* net is the net with all its cells unlocked; a *loose* net is the net with all locked cells in exactly one block and at least one free cell in another block; a *locked* net is the net with locked cells in two or more blocks. Our *loose* net removal process (LR) is to prevent as many *loose* nets from becoming locked as possible during the move of cells, so that we can remove more loose nets from the cutset. The *stable* net is the net that remains in the cutset throughout the entire *run* (where a *run* consists of a number of *passes*). The experiments for some industrial circuits showed that more than 80% of the nets in the final cutset are *stable* nets when FM algorithm was applied [ShNK95]. This means that during FM partitioning process, 80% of the nets were never moved from the cutset since the initial partitioning solution had been generated. From this observation, Shibuya et al. [ShNK95] proposed a hill-climbing method called Stable Net Transition (SNT) to remove some *stable* nets from the cutset. We combined *loose* net removal process (LR) with SNT and developed LSR algorithm. The LSR algorithm is very powerful in removing *loose* and *stable* nets from the cutset. Unlike Dutt and Deng's [DuDe96] CLIP/CDIP algorithm, our LSR algorithm does not need to embed lookahead ability, since it reduced the frequency of tie-breaking situation significantly.

Our research also showed that prior hierarchical clustering can be effectively incorporated into partitioning algorithm to improve the solution quality, even if the cluster removal technique is applied in the partitioning algorithm. In our hierarchical MFFS clustering algorithm, we generalized the MFFC decomposition method [CoDi93, CoLB94, CoXu95] from combinational circuits to general sequential circuits. Our MFFS clustering algorithm can still maintain the natural clustering property as in MFFC clustering. However, it overcomes two limitations in the MFFC clustering algorithm: One is that the MFFS clustering algorithm can handle cycles in the netlist naturally, while MFFC was designed for clustering combinational circuits. The other one is that our MFFS algorithm allows multiple outputs for a cluster so that it can produce more natural clusters and makes hierarchical clustering possible, while MFFC clustering can allow only one output signal for each cluster, which results in many small clusters.

Our partitioning algorithm FM-LSRb begins with the clustered circuit. Next, it focuses on removing *loose* nets from the cutset through a number of passes until there exists no more decrease in the cutsize. Then, we start a hierarchical boundary declustering process which decomposes clusters that straddle on the cut line for additional cutsize reduction. Finally, we detect and remove *stable nets* that have remained in the cutset throughout the entire partitioning process using a hill-climbing global optimization approach [ShNK95].

The FM-LSRb algorithm has achieved the best cutsize result among all the iterative improvement based partitioning algorithms reported in the literature. In particular, it outperforms GMetis [AlHK96], LA3-CDIP [DuDe96], and CLIP-PROP$_f$ [DuDe96] by 9.7%, 12.7%, and 7.1%, respectively. It also outperforms the state-of-the-art non-iterative improvement partitioning algorithm FBB [YaWo94] and Paraboli [RiDJ94]

design strategy has to be used to make VLSI layout more manageable. Circuit partitioning is crucial in top-down hierarchical VLSI layout.

The existing partitioning algorithms in the literature can be classified into two-way partitioning (bipartitioning) algorithms and multiway partitioning algorithms. The bipartitioning algorithms include the iterative improvement methods, the analytical methods, the min-cut based method, and the net-based partitioning method. Some of the best known iterative improvement based partitioning methods include the Kernighan-Lin (KL) algorithm [KeLi70], the Fiduccia and Mattheyses (FM) algorithm [FiMa82], the FM algorithm with lookahead scheme [Kr83], and the simulated annealing approach [KiGV83, GrSu84]. The analytical methods use a linear placement formulation with either the quadratic wirelength objective function, which is solved by computing the second smallest eigenvector of the Laplacian matrix of the given circuit [Ba82, Bo87, DoHo73, HaKa92, AlYa95], or a linear wirelength objective function, which is solved by an iterative method in [RiDJ94, LiLC95]. The min-cut based method uses the maximum flow algorithm to compute a series of minimum cuts in the given circuit in order to obtain an area-balanced cut with small cut size [YaWo94]. The net-based partitioning approach first computes a bipartitioning of the nets, and then transforms the net partitioning solution into a module partitioning solution [CoHK92, CoLS96].

The multiway partitioning algorithms include the recursive bipartitioning algorithm by Kernighan and Lin [KeLi70], a generalization of the FM algorithm with lookahead by Sanchis [Sa89], the primal-dual algorithm [YeCL91], and a generalization of the graph spectral-based partitioning method to multiway ratio-cut by Zien et al. [ZiSC96].

To reduce the computational complexity for partitioning very large circuits, clustering based partitioning methods have been introduced. In this approach, clusters are identified and collapsed, and the resulting clustered circuit is partitioned using existing partitioning methods. Clustering methods include random-walk clustering [CoHK91,HaKa92], multicommodity-flow based clustering [YeCL92], clique-based clustering [CoSm93], geometric embedding with min-diameter clustering [AlKa93], and clustering based on maximum fanout-free cones (MFFC's) [CoDi93, CoLB94, CoXu95]. Partitioning with module replication [HwGa92, KrNe91] and the communication-complexity based partitioning method [BeSa93] have also been proposed to further reduce the amount of interconnections. A comprehensive survey of various techniques in circuit partitioning and clustering can be found in [AlKa95].

Since the iterative improvement based partitioning method is easy to handle various partitioning constraints, and accommodate various module sizes, it has been studied extensively. Recently, Dutt and Deng [DuDe96] introduced an effective cluster-removal technique into their iterative improvement partitioning algorithm which can prevent big clusters from being cut. They pointed out that FM type of iterative improvement partitioning algorithms can efficiently remove small clusters from the cutset, but is very likely to lock bigger clusters in the cutset. In order to overcome this problem, they divided the cell gain into *initial* gain, calculated before the cell movement, and the *updated* gain, generated from the cell movement afterwards. By focusing on the updated gain when choosing cells to move, they were very successful in removing big clusters from the cutset without any prior clustering. Their method outperforms Paraboli [RiDJ94] and MELO [AlYa95] substantially.

Our research revealed that cluster removal can be accomplished more efficiently by focusing on removing

Large Scale Circuit Partitioning With Loose/Stable Net Removal And Signal Flow Based Hierarchical Clustering

Jason Cong, Peter Li, Sung-Kyu Lim, Toshiyuki Shibuya[1] and Dongmin Xu
UCLA Department of Computer Science, Los Angeles, CA 90024
[1]Fujitsu CAD Laboratory, Kawasaki, Japan

Abstract

In this paper, we present an efficient iterative improvement based partitioning algorithm, called FM-LSRb, that combines a signal flow based hierarchical clustering algorithm using Maximum Fanout Free Subgraph (MFFS) technique with a *loose* and *stable* net removal partitioning approach (LSR). The MFFS clustering approach generalized existing MFFC decomposition method from combinational circuits to general sequential circuits, which enable us to handle cycles in circuits naturally, and make hierarchical clustering and declustering possible. We also studied the properties of the nets that straddle the cutline carefully, and introduced the concepts of the *loose* nets and *stable* nets. Our FM-LSRb algorithm begins with the hierarchical MFFS clustering based on signal direction, logical dependency and connectivity. Then, it focuses on removing *loose* and *stable* nets from the cutset of the clustered netlist. A hierarchical declustering process was embedded in the partitioning algorithm, which was used to decompose clusters that straddled on the cutline for additional cutsize reduction. The FM-LSRb algorithm has achieved the best cutsize result among all the iterative improvement based partitioning algorithms reported in the literature. In particular, it outperforms GMetis [AlHK96], LA3-CDIP [DuDe96], and CLIP-PROP$_f$ [DuDe96] by 9.7%, 12.7%, and 7.1%, respectively. It also outperforms the state-of-the-art non-iterative improvement partitioning algorithm FBB [YaWo94] and Paraboli [RiDJ94] by 17.2% and 30.0%, respectively.

1 Introduction

Circuit partitioning divides a given large circuit into a collection of smaller subcircuits, called blocks, to minimize the number of connections among the subcircuits, subject to the balanced area constraint. The partitioning problem is becoming more important as VLSI technology reaches submicron device dimensions. The feature size in next generation of VLSI ASIC will be below 0.25 micron and a single VLSI chip can contain over 10 million gates. As a result, the circuit layout problem becomes much more difficult. The well-accepted simulated annealing based or analytical placement methods [SeLe87, SwSe95, KlSJ91, SrCK91, TsKH88] become infeasible when applied to the flat netlist, since they will require prohibitive amount of computation time and/or memory to produce a satisfactory solution. Therefore, the top-down hierarchical