Large Scale Circuit Partitioning With Loose/Stable Net Removal And Signal Flow Based Clustering [1]

Jason Cong, Honching Peter Li, Sung Kyu Lim, Toshiyuki Shibuya* and Dongmin Xu
UCLA Department of Computer Science, Los Angeles, CA 90095
*Fujitsu Laboratories LTD., Kawasaki, Japan

In this paper, we present an efficient Iterative Improvement based Partitioning (IIP) algorithm called LSR/MFFS, that combines signal flow based Maximum Fanout Free Subgraph (MFFS) clustering algorithm with Loose and Stable net Removal (LSR) partitioning algorithm. The MFFS algorithm generalizes existing MFFC decomposition method from combinational circuits to general sequential circuits in order to handle cycles naturally. We also study the properties of the nets that straddle the cutline carefully, and introduce the concepts of the *loose* and *stable* nets as well as effective ways to remove them out of the cutset. The LSR/MFFS algorithm first applies LSR algorithm to clustered netlist generated by MFFS algorithm for global-level cutsize optimization and then declusters netlist for further cutsize refinement.

As a result, the LSR/MFFS algorithm has achieved the best cutsize result among all the bipartitioning algorithms published in the literatures with very promising runtime performance. In particular, it outperforms the recent state-of-the-art IIP algorithms LA3-CDIP, CLIP-PROP$_f$ [8], Strawman [12], hMetis-FM [13], and MLc [1] by 17.4%, 12.1%, 5.9%, 3.1%, and 1.9%, respectively. It also outperforms the state-of-the-art non-IIP algorithms Paraboli [17], FBB [19], and PANZA [16] by 32.0%, 21.4%, and 1.4%, respectively.

1 Introduction

Circuit partitioning divides a given circuit into a collection of smaller subcircuits to minimize the number of connections among the subcircuits, subject to the area balance constraint. The circuit partitioning problem becomes more important as VLSI technology reaches submicron device dimensions; a single VLSI chip can contain over 10 million gates for the 0.25 micron technology and beyond. As a result, the hierarchical layout strategy using the divide-and-conquer technique is indispensable in order to make the VLSI layout problem tractable.

The existing circuit partitioning algorithms in the literature can be roughly classified into two classes; constructive methods, such as the spectral-based methods [11, 3] and the network flow-based method [19], and iterative improvement methods (also referred as group migration or move based methods). In practice, iterative improvement based partitioning (IIP) algorithms have been used extensively due to the following advantages over the other approaches; (i) area balance constraint, pre-assignment of cells, and non-uniformity in cell sizes can be easily accommodated, (ii) one can easily control the runtime vs cutsize trade-off by controlling the number of iterations. Some of the best known methods include the Kernighan & Lin (KL) algorithm [14], the Fiduccia & Mattheyses (FM) algorithm [9], and Krishnamurthy's lookahead scheme [15]. To reduce the computational complexity for partitioning large-scale circuits, clustering methods have been introduced. In this case, clusters are first identified and collapsed, and the resulting clustered circuit is partitioned using existing partitioning methods. Among many promising studies on circuit clustering methods, we focus on the extension of signal flow based Maximum Fanout Free Cone (MFFC) approach [4, 6, 7]. A comprehensive survey of various techniques in circuit partitioning and clustering can be found in [2].

In this paper, we present a large-scale IIP algorithm named LSR/MFFS. It integrates three individual algorithms to accomplish both global and local-level cutsize optimization; Loose net Removal (LR), existing Stable Net Transition [18] (SNT), and Maximum Fanout Free Subgraph (MFFS) algorithm. The LR and SNT algorithms focus on nets instead of cells to reduce the cutsize by attempting to remove two special kinds of nets; *loose* and *stable* net (to be defined in the subsequent sections). Our study shows that LR is effective in dynamically identifying clusters during cell movements, while SNT enables the partitioning algorithm to jump out of local minima efficiently. The intergration of these two net removal schemes results in a very powerful IIP algorithm named Loose and Stable net Removal (LSR).

In order to reduce the runtime and prune non-optimal soltions from the solution space, we developed signal flow based MFFS clustering algorithm that promotes simultanoeus movement of logically dependent cells during the cell moves. Our LSR/MFFS algorithm first clusters the circuit using the MFFS clustering algorithm, then applies the LSR algorithm to the clustered circuit, and finally declusters the circuit and applies LSR algorithm again for further cutsize refinement. Our experiments indicate that LSR/MFFS partitioning algorithm has achieved the best cutsize results with very promising runtime performance among all the partitioning algorithms reported in the recent literature.

The remainder of the paper is organized as follows. Section 2 presents the LSR based partitioning algorithm. Section 3 presents signal flow based MFFS clustering algorithm. Section 4 provides experimental results. Section 5 concludes the paper with our ongoing research.

[1]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. Many thanks are due to the authors of hMetis [13] for their helpful discussions on runtime performance.

2 LSR based Partitioning Algorithm

2.1 Review of IIP Algorithms

An *iterative improvement* partitioning (IIP) algorithm such as FM [9] 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* and precomputed gain of entire cell, IIP algorithm selects the maximum gain cell c that satisfies the size constraint to move. A cell is labeled *free* if it has not been moved so far and *locked* otherwise. The *neighbors* of c are the free cells of the nets that are incident to c. After c is moved and locked, gain of its neighbors is updated for the next move. This cell move continues until all the free cells that do not violate the area constraint are moved, and 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 is attempted until there is no further improvement in terms of cutsize. In this case, these multiple passes constitute a *run*.

IIP algorithm usually adopts a special data structure called *bucket* that supports fast update and retrieval of cell gains. Recently, the LIFO bucket [10] scheme is proposed to ensure that the cells with the most recently updated gain to be chosen first among other cells with the same gain, providing different perspective in breaking ties. This scheme is designed to promote the *locality* of cell moves where all the neighbors of the currently moved cell are chosen to move subsequently. However, locality can easily be disrupted by the update of other cells with higher gain values under this scheme. Dutt and Deng [8] proposed a more efficient way of promoting the locality. They viewed the cell gain as the sum of *initial* and *updated* gain, where the former refers to the cell gain computed before any move, and the latter refers to the sum of dynamically updated gain values from the cell move afterwards. Their strategy is based on the observation that relying only on the updated part of the gain in choosing the next cell encourages the neighboring cells to move subsequently and thus establishes the locality. They also noted that this locality promotes closely connected components in the circuit (= clusters) that straddle the cutline to be removed out of the cutset and thus establishes *cluster removal*. As a result, their algorithm CLIP/CDIP achieved the best cutsize result among all the published IIP algorithms prior to their work. However, their strategy is not free from the frequent tie-breaking situation, still relying on LIFO bucket and lookahead capability such as LA3 [15].

2.2 LSR Partitioning Algorithm

We present in this section stronger yet simpler form of cluster removal strategy by gradually increasing the cell 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 maintains the locality. It also reduces the frequency of tie-breaking situation significantly 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, then present a modified cell gain computation of Loose-net Removal (LR) algorithm and its enhancement with Stable Net Transition (SNT) [18]. The following discussion focuses on bipartitioning case, but LSR can be easily extended to handle multiway partitioning.

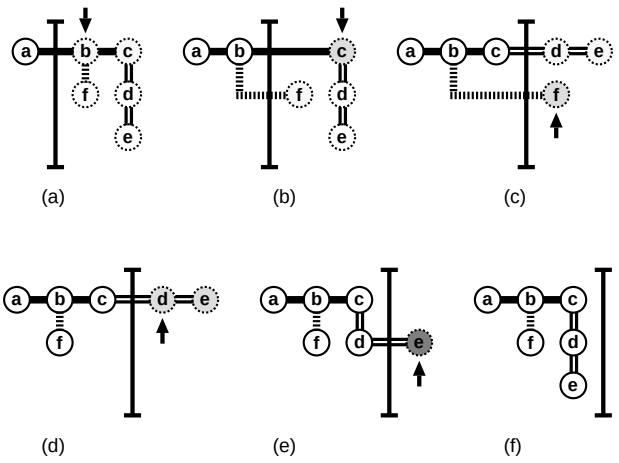

Figure 1: Removing 3 nets $\{a, b, c\}$, $\{b, f\}$, and $\{c, d, e\}$ with LR. Arrow points to cells chosen to move, and dotted and solid circle denote free and locked cell, respectively. Shade indicates cells with LR's cell gain increase, which darkens as the increase accumulates.

2.2.1 Loose Net Removal

During each pass of IIP algorithm, *free* net contains only free cells, and a *locked* net has locked cells in two blocks. A net is defined to be *loose* if it has locked cells in exactly one block and at least one free cell in the other block. The *anchor* block of a loose net n is defined as the block that contains the locked cells of n, and the other block that has one or more free cells of n is called *tail* block. Once a free cell c of a free net n is moved and locked into a block B, net n immediately becomes loose with B being its anchor. For example, in Figure 1-a, net $\{a, b, c\}$ is a loose net with the left block being its anchor. But if any of the free cells of the loose net in its tail block, say b or c, is moved and locked at a non-anchor block, the net will remain locked into the cutset thereafter during the entire pass. In other words, every net in the final cutset follows the $free \rightarrow loose \rightarrow locked$ state transition, 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 focus on the free cells of a loose net in the tail block while they remain free. It intentionally increases the cell gains associated with moving those cells to its anchor block so that they are more likely to be selected during the subsequent moves. That is, our primary goal is to prevent the corresponding loose net from becoming locked into the cutset. A natural question arises on what to do with the free cells of a loose net located *inside* the anchor. We do not restrict the movement of these cells to non-anchor block since they can participate in removal of other loose nets. In case these cell moves fail to overcome the local minima and degrade the solution quality, conventional FM-type backtrack can always reverse these moves. After each cell move, LR considers free cells of all loose nets that are incident to the currently moved cell, promoting these cells to move to their anchor blocks by increasing their gains. Therefore, LR is effective in maintaining the locality of cell movement and thus establishing the cluster removal effect.

2.2.2 Cell Gain Increase Function

Removal of the given initial loose net causes additional loose nets to form subsequently, necessitating rules to set priority in removing these nets. The magnitude of cell gain increase due to a given loose net n is then 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 max_net_size denotes the size of the largest net in the given netlist, $size(n)$ denotes the number of cells of n, and $deg(c)$ denotes the number of nets incident to cell c. L_n denotes the set of all locked cells of n in the anchor, and F_n denotes the set of all free cells of n in the tail. 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.

```
LR()
while (there exists free cell)
   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)
            if (f.gain + incr(n) < T)
               f.gain = f.gain + incr(n);
endwhile
```

Figure 2: Loose Net Removal Algorithm (single pass)

The LR algorithm based on the modified gain computation is described in Figure 2. Some important points to note in regards to cell gain increase of LR are as follows; First, the same amount of increase is added uniformly to all free cells of loose nets, and the increases are accumulated until the loose net is completely removed to a single block or possibly locked into the cutset. Second, a threshold value T limits the total accumulated value of cell gain, which enables us to adopt conventional array-based bucket structure [9]. Note that the only reason for the gain increase is to ensure that the target cells are chosen to move prior to other cells. Our study shows that a fairly small threshold value 100 can accomplish this goal without a major performance degradation. Third, the conventional cell gain update is still applied to the neighbors before the increase. This is necessary while choosing the next good initial seed for LR after completing removal of the current cluster. Lastly, the increase eliminates the necessity of special tie-breaking strategy due to the expanded cell gain range. The conventional cell gain ranges from $-p$ to p, where p represents the maximum number of nets incident to a single cell. Since the entire cells are distributed into this short range, most other IIP algorithms require special tie-breaking strategies such as LIFO bucket [10] or lookahead capability [15]. However, LR reduces the frequency of tie-breaking situation significantly since the rank of cells in the bucket is determined according to the increased gain values.

Our experiments show that this simple strategy works surprisingly well; LR alone ourperforms LIFO-FM, LIFO-LA3, LIFO-LA3-CLIP, and LIFO-LA3-CDIP by 8.2% to 54.4% improvement in terms of minimum cutsize of 20 runs with less CPU time (interested reader is referred to [5] for details). This convinces us that our simple cluster removal scheme focusing on the free cells of loose nets is very effective without any kind of tie-breaking enhancement. Figure 1 shows an example of removing three nets $n_1 = \{a, b, c\}$, $n_2 = \{b, f\}$, and $n_3 = \{c, d, e\}$ from the cutset with LR. First of all, LR increases the gain of b and c since n_1 is a loose net (1-a). Since LR favors shorter nets, n_2 will be chosen first to be removed by moving b (1-b). Then, LR first finishes removing n_1 (1-c) prior to n_2 (1-d). Finally, n_3 is removed (1-f). This example shows only a single direction of LR, but the similar process can be performed in the other direction, enabling us to balance the cut.

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 [18] observed that more than 80% of the nets in the final cutset are stable, and these nets trap FM 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 algorithm to systematically explore more local minima yet still with a reasonable CPU time.

After a run of FM algorithm terminates, the initial and the 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 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 nets are processed or no more move is possible. Then, another run of FM is performed using the outcome of the stable net removal as its initial partition. Usually the cutsize increases right after removing the stable net, but such hill-climbing is not so costly because SNT promotes higher rate of convergence and shorter duration of runs due to the fact that moving the stable nets only would result in a minor perturbation of the current configuration compared to an entirely new random initial configuration.

The idea of stable net removal can be combined naturally with loose net removal. LR dynamically removes nets in the cutset during the cell moves, whereas SNT statically removes nets from the cutset at the end of cell moves. Integrating these two net removal schemes results in a powerful partitioner named Loose and Stable net Removal (LSR).

3 MFFS Based Clustering Algorithm

Efficient clustering algorithms are important 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 [4] from combinational circuits to general sequential circuits.

The MFFC based clustering algorithm has been reported to provide natural clustering solution for combinational circuits [4, 6]. A modified MFFC algorithm for sequential circuits has been presented in [7], but the capability of this algorithm to detect directed cycles is limited. In general, the MFFC algorithm is mainly applied to combinational

circuits. In the following section, we introduce a clustering method based on the Maximum Fanout Free Subgraph (MFFS), which can overcome the restriction of MFFC clustering.

3.1 Definition of MFFS

We first rewrite the definition of FFC and MFFC as follows. One can verify that our new definition is equivalent to that in [4].

Definition 1 (FFC & MFFC) *For a given node v in a combinational circuit,*

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

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 2 (FFS & MFFS) *For a given node v in a sequential circuit,*

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

3.2 MFFS Construction Algorithm

For a given netlist NL and a node v, the MFFS cluster rooted at v can be obtained by using the following procedure:

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 j is a fanin of node i.
2. Cut all the fanout edges of the root node v; search all other nodes in graph $G(N, E)$ starting from the primary outputs (POs) of the netlist. The nodes in $G(N, E)$ that are not traversed belong to the $MFFS_v$.

This algorithm is named as the MFFS construction algorithm.

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 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. Our MFFS clustering algorithm works as follows:

For a given netlist NL, let $roots = \{\text{all POs of } NL\}$. Then, we take out a node $v \in roots$ and use the MFFS construction algorithm to construct $MFFS_v$. This process is repeated until $roots$ is empty. Then we remove all the currently constructed MFFS clusters from NL, and obtain a reduced netlist NL' whose POs are the input nodes to the removed MFFS clusters. We repeat the same procedure for the new netlist NL' recursively until all cells in netlist NL are grouped into MFFS clusters. One can refer to [5] for an example, illustrating how the algorithm works.

Theorem 2 *The time complexity of the MFFS clustering algorithm is $O(|N| \times (|N| + |E|))$ in the worst case, where $|N|$ is the number of nodes and $|E|$ is the number of edges in the directed graph, G(N, E).*

3.4 Speedup of MFFS Algorithm

Although our MFFS clustering algorithm can find good clusters, its runtime can be long for large cicuits due to the fact that each MFFS construction requires the search of the entire netlist. Since in practice we usually have a maximum size limit to each cluster, we have developed a heuristic version of MFFS clustering algorithm that only searches a subset of the circuit every time. We call this algorithm MFFSd(h), where h indicates the depth of the circuit to be searched. Given a node v, this algorithm constructs an approximate of $MFFS_v$ as follows:

1. Perform breadth-first search from node v of depth h in $G(N,E)$ and put all encountered nodes in set SC_v.
2. For every node in SC_v, if its fanout nodes are not in SC_v, put them into set PPO_v (called pseudo POs).
3. Apply the MFFS Construction Algorithm, described in Section 3.2, on the subcircuit formed by SC_v and PPO_v with all nodes. in PPO_v being considered as POs. We can then obtain a cluster rooted at node v.
4. repeat (1) until all nodes has been clustered.

Since only a small number of cells need to be searched to find a cluster in step (3) when h is set to be small, we can consider that this step takes constant time. Therefore, the resulting MFFSd(h) clustering algorithm requires only $O(|N| + |E|)$ time.

4 Experimental Result

We have implemented our LSR/MFFS and MFFSd(h) algorithm, compiled with gcc v2.4, and tested them on SUN SPARC5-85. Table 1 lists some characteristics of benchmark circuits used in our experiments. Selection of test circuits is limited to those with signal direction, but most of them are used in common among other published papers. The bipartitioning results are obtained using the minimum cutsize of 20 runs under 45-55% area balance criterion and real cell sizes. The runtime is measured in seconds.

4.1 Cutsize Reduction Trend

The experimental result summerized in Table 1 shows the cutsize reduction trend starting from the basic FM to our most enhanced LSR/MFFS. We tested FM, SNT, LR, and LSR with and without MFFS clustering/declustering. The following observations are made from this set of experiment regarding various enhancements developed in this paper; (i) Signal flow based MFFS clustering indeed helps to reduce the cutsize and runtime. This is evident from the comparison between FM vs FM/MFFS, SNT vs SNT/MFFS, etc. (ii) LR is the most effective contributor in terms of cutsize reduction. This is revealed from the comparison between FM vs LR and FM/MFFS vs LR/MFFS. (iii) SNT provides good initial partition and improves both cutsize and runtime, when combined with LR. This is apparent from the comparison between FM vs SNT, LR vs LSR, FM/MFFS vs SNT/MFFS and LR/MFFS vs LSR/MFFS.

Benchmark Circuits					No Clustering Based				MFFS Clustering Based			
name	# cell	AVR	CT1	CT2	FM	SNT	LR	LSR	FM	SNT	LR	LSR
s1423	619	1.0	0.9	1.9	12	14	13	13	12	12	12	12
sioo	664	4.6	2.3	2.6	25	25	25	25	25	25	25	25
s1488	686	1.0	0.9	1.4	48	45	44	42	42	42	43	43
balu	801	12.9	2.0	3.8	27	27	27	27	27	27	27	27
prim1	833	6.7	3.2	5.9	43	42	42	42	43	42	42	43
struct	1952	2.5	17.9	12.1	42	45	33	33	39	52	33	33
prim2	3014	6.7	34.8	22.1	174	167	121	122	133	122	131	119
s9234	5866	1.0	36.7	9.7	49	54	41	43	42	43	41	40
biomed	6514	4.5	87.8	30.5	83	83	83	83	100	84	84	84
s13207	8772	1.0	73.9	14.1	90	74	75	70	91	65	73	61
s15850	10470	1.0	84.9	17.6	113	73	73	65	72	70	44	43
s35932	18148	1.0	420.8	32.1	100	128	45	44	48	76	45	44
s38584	20995	1.0	565.1	44.5	52	52	49	47	52	52	47	47
avq.sm	21918	4.5	1287.3	116.1	321	378	135	139	273	172	127	127
s38417	23949	1.0	452.2	45.1	176	112	53	55	119	147	51	50
avq.lg	25178	4.5	1473.2	90.2	491	380	145	131	251	168	127	127
Total	-	-	4543.9	449.7	1846	1699	1004	981	1369	1261	952	925
Time	-	-	-	-	214.6	100.9	326.0	279.0	139.1	102.1	206.5	162.6

Table 1: Characterization of the benchmark circuits and cutsize reduction from various enhancements. AVR stands for Area Variation Ratio (= max cell area/min cell area). Clustering time using MFFS and MFFSd(h) algorithms are reported under CT1 and CT2 column, respectively. The total CPU time of single run of each partitioning algorithm is listed under Time row.

4.2 Comparison to Published Results

Table 2 summerizes the comparisons of LSR/MFFS with other recently published state-of-the-art partitioning algorithms in terms of cutsize and runtime. We reported the sum of total elapsed CPU time for 10 runs of partitioning (and clustering, if applicable) 13 circuits under IIP algorithms.

IIP algorithms in comparison include LA3-CDIP, CLIP-PROP$_f$ [8], Strawman [12], hMetis-FM$_{20}$ [13], and MLc-100 [1]. The experiment shows that LSR/MFFS consistently outperforms non-clustering based algorithms LA3-CDIP and CLIP-PROPf by 17.4% and 12.1%, respectively, in terms of cutsize. In addition, LSR/MFFS outperforms multi-level clustering based Strawman, hMetis-FM$_{20}$ and MLc-100 by 5.9%, 3.1%, and 1.9%, respectively. The comparison of runtimes to other IIP algorithms except hMetis also reveals the performance superiority of LSR/MFFS.

Non-IIP algorithms in comparison include PANZA [16], Paraboli [17], and FBB [19]. LSR/MFFS again consistently outperforms these algorithms by 1.4%, 32.0%, and 21.4%, respectively. PANZA omitted runtime in their report, and the average runtime for each run of FBB tested on SPARC10 sums to 921.3 sec for 7 circuits tested, whereas LSR/MFFS spends only 79.8 sec. In overall, LSR/MFFS has achieved the best cutsize result among all the partitioning algorithms published in the literature with very promising runtime.

5 Conclusion & Ongoing Work

In this paper, we introduced the notion of loose net and presented the LR (Loose net Removal) algorithm. LR was then enhanced with existing SNT (Stable Net Transition) scheme [18] for eliminating hard-to-remove nets from the cutset. We also generalized the existing MFFC decomposition method to come up with signal flow based MFFS (Maximum Fanout Free Subgraph) clustering algorithm. The LSR/MFFS algorithm first applies LSR algorithm to clustered netlist generated by MFFS algorithm for global-level cutsize optimization and then declusters netlist for further cutsize refinement. As a result, LSR/MFFS partitioning algorithm has achieved the best cutsize results among all the partitioning algorithms reported in the recent literature.

Our ongoing research includes (i) extension of MFFS algorithm to support multi-level cluster hierarchy as well as various runtime improvement schemes, (ii) extension of LR to handle multiway partitioning efficiently, and (iii) development of a mincut-based placement algorithm exploiting the promising performance of LSR/MFFS.

REFERENCES

[1] C. J. Alpert, D. Huang, and A. B. Kahng. "Multilevel circuit partitioning". In *Proc. ACM/IEEE Design Automation Conf.*, pages 530–533, 1997.

[2] C. J. Alpert and A. B. Kahng. "Recent directions in netlist partitioning: a survey". *Integration, the VLSI Journal*, pages 1–81, 1995.

[3] C. J. Alpert and S. Z. Yao. "Spectral partitioning: The more eigenvectors, the better". In *Proc. ACM/IEEE Design Automation Conf.*, pages 195–200, 1995.

[4] J. Cong and Y. Ding. "On area/depth trade-off in LUT-based FPGA technology mapping". In *Proc. 30th ACM/IEEE Design Automation Conf.*, pages 213–218, 1993.

[5] J. Cong, P. Li, S. K. Lim, T. Shibuya, and D. Xu. "Large scale circuit partitioning with loose/stable net removal and signal flow based hierarchical clustering". Technical Report 970005, CS Dept. of UCLA, 1997.

	IIP Algorithms						Non-IIP Algorithms		
	CDIP	$PROP_f$	Straw	hMetis	MLc	**LSR/MFFS**	PANZA	Paraboli	FBB
circuit	[8]	[8]	[12]	[13]	[1]	-	[16]	[17]	[19]
s1423	-	-	-	-	-	**12**	15	16	13
sioo	-	-	-	-	-	**25**	25	45	-
s1488	-	-	-	-	-	**43**	44	50	-
balu	27	27	27	27	27	**27**	27	41	-
prim1	52	51	49	50	47	**43**	-	-	-
struct	36	33	33	33	33	**33**	33	40	-
prim2	152	152	143	145	139	**119**	-	-	-
s9234	44	42	42	40	40	**40**	40	74	70
biomed	83	84	83	83	83	**84**	83	135	-
s13207	70	71	57	55	55	**61**	66	91	74
s15850	67	56	44	42	44	**43**	44	91	67
s35932	73	42	47	42	41	**44**	43	62	49
s38584	47	51	49	47	47	**47**	47	55	47
avq.sm	148	144	131	130	128	**127**	-	-	-
s38417	79	65	53	51	49	**50**	49	49	58
avq.lg	145	143	140	127	128	**127**	-	-	-
Total1	1023	961	898	872	861	**845**	-	-	-
Total2	-	-	-	-	-	**509**	516	749	-
% Imp	17.4	12.1	5.9	3.1	1.9	-	1.4	32.0	21.4
Time	5817	5611	12577	***[1]	3455	**2054**	-	24619	-

Table 2: Comparison of LSR/MFFS to other published algorithms. Total1 is based on 13 common test circuits under IIP algorithms, whereas Total2 is based on 12 common test circuits under non-IIP algorithms. Our runtime includes MFFSd(h) clustering time from Table 1.

[6] J. Cong, Z. Li, and R. Bagrodia. "Acyclic multi-way partitioning of boolean networks". In *Proc. ACM/IEEE 31st Design Automation Conf.*, pages 670–675, 1994.

[7] J. Cong and D. Xu. "Exploiting signal flow and logic dependency in standard cell placement". In *Proc. Asia and South Pacific Design Automation Conf.*, pages 399–404, 1995.

[8] S. Dutt and W. Deng. "VLSI circuit partitioning by cluster-removal using iterative improvement techniques". In *Proc. Int'l Conf. on Computer-Aided Design*, pages 194–200, 1996.

[9] C. Fiduccia and R. Mattheyses. "A linear time heuristic for improving network partitions". In *Proc. ACM/IEEE Design Automation Conf.*, pages 175–181, 1982.

[10] L. Hagen, D. Huang, and A. B. Kahng. "On implementation choices for iterative improvement partitioning algorithms". In *Proc. ACM/IEEE European Design Automation Conf.*, 1995.

[11] L. Hagen and A. B. Kahng. "A new approach to effective circuit clustering". In *Proc. Int'l Conf. on Computer-Aided Design*, pages 422–427, 1992.

[12] S. Hauck and G. Borriello. "An evaluation of bipartitioning techniques". *submitted to IEEE Trans. on Computer-Aided Design*, 1996.

[13] G. Karypis and V. Kumar. "Multilevel hypergraph partitioning : Application in VLSI domain". In *Proc. ACM/IEEE Design Automation Conf.*, pages 526–529, 1997.

[14] B. Kernighan and S. Lin. "An efficient heuristic procedure for partitioning of electrical circuits". *Bell System Technical Journal*, 1970.

[15] B. Krishnamurthy. "An improved min-cut algorithm for partitioning VLSI networks". *IEEE Trans. on Computers*, pages 438–446, 1984.

[16] J. Li, J. Lillis, and C. K. Cheng. "Linear decomposition algorithm for VLSI design applications". In *Proc. ACM/IEEE Design Automation Conf.*, pages 223–228, 1995.

[17] B. M. Riess, K. Doll, and F. M. Johannes. "Partitioning very large circuits using analytical placement techniques". In *Proc. ACM/IEEE 31st Design Automation Conf.*, pages 646–651, 1994.

[18] T. Shibuya, I. Nitta, and K. Kawamura. "SMINCUT: VLSI placement tool using min-cut". *Fujitsu Scientific & Technical Journal*, pages 197–207, 1995.

[19] H. Yang and D. F. Wong. "Efficient network flow based min-cut balanced partitioning". In *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 50–55, 1994.

[1] A direct comparison between hMetis and LSR/MFFS is not possible since hMetis used SGI R10000 200MHz for collecting runtime. However, private communication between the authors indicates that hMetis is about 1.4 times faster.