To appear in Proc. IEEE/ACM International Conference on CAD, 1996

VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques

Shantanu Dutt[1] and Wenyong Deng [2]
[1] Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455, USA
dutt@ee.umn.edu
[2]LSI Logic Corporation, Milpitas, CA 95035, USA
wydeng@lsil.com

Abstract

Move-based iterative improvement partitioning methods such as the Fiduccia-Mattheyses (FM) algorithm [3] and Krishnamurthy's Look-Ahead (LA) algorithm [4] are widely used in VLSI CAD applications largely due to their time efficiency and ease of implementation. This class of algorithms is of the "local improvement" type. They generate relatively high quality results for small and medium size circuits. However, as VLSI circuits become larger, these algorithms are not so effective on them as direct partitioning tools. We propose new iterative-improvement methods that select cells to move with a view to moving clusters that straddle the two subsets of a partition into one of the subsets. The new algorithms significantly improve partition quality while preserving the advantage of time efficiency. Experimental results on 25 medium to large size ACM/SIGDA benchmark circuits show up to 70% improvement over FM in cutsize, with an average of per-circuit percent improvements of about 25%, and a total cut improvement of about 35%. They also outperform the recent placement-based partitioning tool Paraboli [13] and the spectral partitioner MELO [14] by about 17% and 23%, respectively, with less CPU time. This demonstrates the potential of iterative improvement algorithms in dealing with the increasing complexity of modern VLSI circuitry.

1. Introduction

The essence of VLSI circuit partitioning is to divide a circuit into a number of subcircuits with minimum interconnections between them. This can be accomplished by recursively partitioning a circuit into two parts until we reach the desired level of complexity. Thus two-way partitioning is a basic problem in circuit partitioning and placement.

Kernighan and Lin [1] proposed the well-known KL algorithm for graph partitioning. The KL algorithm starts with a random initial two-way partition and proceeds by swapping pair of cells iteratively. Schweikert and Kernighan [2] extended KL to hypergraphs so that it can partition actual circuits. Fiduccia and Mattheyses [3] modified this to a reduced-complexity algorithm (FM) of time linear in the number of pins in the circuit. This is done by moving one cell at a time and using an efficient bucket data structure. Using an efficient best-pair search strategy, Dutt [11] significantly improved the complexity of the original KL algorithm, i.e., with pair swaps, from $\Theta(n^2 \log n)$ to $\Theta(ed)$, where n is the number of nodes, e the number of edges, and d the maximum node degree; empirical results show that it is as fast as FM and gives better results. Krishnamurthy [4] enhanced FM by adding higher level lookahead gains and improved the results for small circuits, while Hagen et al. [15] investigated implementation and tie-breaking techniques for improving the performance of FM-type algorithms. A number of clustering-based, i.e., bottom-up, partitioning algorithms [9, 10, 12, 13, 14, 18] have also been proposed and good results have been obtained.

FM and LA are the most commonly used two-way partitioning algorithms largely due to their excellent run times, simple implementations and flexibility. However, this class of iterative improvement algorithms have a common weakness, viz., they only find solutions corresponding to local minima. Because of their iterative improvement nature, they can only evolve from an initial partition through very short-sighted moves. Thus the results strongly depend on the initial partition. In order to get a local minimum that is close to the optimum partition, multiple runs on randomly generated initial partitions are needed. As the circuit size becomes large, the probability of finding a good local minimum in one run will drop significantly. This makes FM an unattractive choice for partitioning large circuits. As will become clear later, FM gives good results on small to medium size circuits, but performs very poorly on large circuits. Some clustering [8] or compaction [18] techniques have been proposed to remedy the above weakness. Very good results have been obtained at the cost of considerable CPU time increases and implementation complexities. In this paper, we will propose a technique that significantly improves the ability of iterative improvement methods like FM and LA for finding good local minima. The new technique pays more attention to the neighbors of moved cells and encourages the successive moves of closely connected cells. This implicitly promotes the move of an entire densely-connected group (a cluster) into one subset of the partition. The large reduction in both the minimum and average cutsize of multiple runs indicates that the new technique is a more robust and stable approach. The increase in implementation complexity and run time is minimal. We also propose a sophisticated extension of this basic technique, which explicitly identifies clusters during the move sequence, and dynamically adapts the move sequence to facilitate the move of clusters into one subset of the partition. Very good results have been obtained at reasonable CPU times.

In the next section, we briefly describe the FM and LA algorithms and point out their shortcomings. Then in Section 3. we present our rationale for the new technique mentioned above, and also propose a cluster-detecting algorithm based on iterative improvement methods. Extensive experimental results are presented in Section 4. along with discussions. Conclusions are in Section 5..

2. Previous Iterative Improvement Algorithms

A circuit netlist is usually modeled by a hypergraph $G = (V, E)$, where V is the set of cells (also called nodes) in the

ICCAD '96
1063-6757/96 $5.00 © 1996 IEEE

circuit, and E is the set of nets (also called hyperedges). We will represent a net n_i as a set of the cells that it connects. A *two-way partition* of G is two disjoint subsets V_1 and V_2 such that each cell $v \in V$ belongs to either V_1 or V_2. A net is said to be *cut* if it has at least one cell in each subset and *uncut* otherwise. We call this the *cutstate* of the net. All the nets that are cut form a set called the *cutset*. The objective of a two-way partitioning is to find a partition that minimizes the size of the cutset (called the *cutsize*). Usually there is a predetermined *balance criterion* on the size of the subsets V_1, V_2, for example, $0.45 \leq |V_i|/|V| \leq 0.55$, where $i = 1, 2$.

The FM algorithm [3] starts with a random initial partition. Each cell u is assigned a gain $g(u)$ which is the *immediate* reduction in cutsize if the cell is moved to the other subset of the partition:

$$g(u) = \sum_{n_i \in E(u)} c(n_i) - \sum_{n_j \in I(u)} c(n_j) \qquad (1)$$

where $E(u)$ is the set of nets that will be immediately moved out of the cutset on moving cell u, $I(u)$ is the set of nets that will be newly introduced into the cutset, and $c(n_i)$ is the weight (cost) of the net n_i.

At each move of FM, the cell with maximum gain value in both subsets is checked first to see if its move will violate the balance criterion. If not, it is chosen as the cell for the current move (the *base cell*). Otherwise, the cell with maximum gain in the other subset is chosen as the base cell. The base cell is then moved to the other subset, "locked"[1], and its gain is inserted in an ordered set S. The gains of all the affected neighbors are updated—a cell v is said to be a *neighbor* of another cell u, if v and u are connected by a common net. The next base cell is chosen in the same way from the remaining "free" (unlocked) cells and the move process proceeds until all the cells are moved and locked. Then all the partial sums $S_j = \sum_{t=1}^{j} g(u_t), 1 \leq j \leq n$, are computed, and p is chosen so that the partial sum S_p is the maximum. This corresponds to the point of minimum cutsize in the entire moving sequence. All the cells moved after u_p are reversed to their previous subset so that the actually moved cells are $\{u_1, \ldots, u_p\}$. This whole process is called a *pass*. A number of passes are made until the maximum partial sum S_p is no longer positive. This is a local minimum with respect to the initial partition $[V_1, V_2]$.

The FM algorithm has been criticized for its well-known shortsightedness [4, 16]—it moves a cell based on the immediate decrease in cutsize. Thus it tends to be trapped in local minima that strongly depend on the initial random partition. We will later point out some other consequences of this shortsightedness that are related to the removal of natural clusters from the cutset.

The FM gain calculation only considers *critical* nets whose cutstate will change *immediately* after the move of the cell. It is conceivable there will be many cells having the same gain value since the gain is bounded above by p_{max} and below by $-p_{max}$, where p_{max} is the maximum degree of a cell. Krishnamurthy has proposed a lookahead gain calculation scheme which includes less critical nets [4]. In addition to FM gain, higher order gains are used to break the ties.

[1]The locking of a moved cell is necessary to prevent thrashing (a cell being moved back and forth) and being trapped in a bad local minimum.

3. Clustering-Based Iterative Improvement Methods

3.1. Case for a Cluster-Oriented Approach

A real VLSI circuit netlist can be visualized as an aggregation of a number of highly connected subcircuits or clusters. This fact has motivated the proposition of many clustering-based algorithms. It is conceivable that there are many levels of clusters with different degrees in the density of their connectivities. A small group of densely interconnected cells may be part of a larger but less densely connected cluster. The goal of our partitioning methods are to determine a cut that goes through the most weakly connected groups.

Iterative improvement algorithms like FM and LA start with a randomly assigned partition that results in a binomial distribution of cells in V_1 and V_2. In such a distribution, the probability of finding r $(r \leq m)$ cells in V_1 and $m - r$ cells in V_2 in some group of m cells is $P(m, r) = \binom{m}{r} p^r q^{m-r}$, where p (q) is the probability of a cell being assigned to V_1 (V_2). For a random partition, $p = q = 0.5$. The probability distribution maximizes at $r = m/2$ with standard deviation $\sigma = \sqrt{m}/2$. For example, a group of 100 cells will have the expected distribution of 50 cells in each subset with a standard deviation of 5 cells. Therefore, for a cluster with a fair number of cells, there is a very high probability that it will initially be cut. Hence in an initial random partition, most clusters will straddle the *cutline*, which is an imaginary line that divides the cells into the two subsets of the partition. This situation is illustrated in Figure 1(a).

For an iterative improvement algorithm to succeed, it must be able to pull clusters straddling the cutline into one subset. It is easy to visualize that there will be many cells in different clusters with similar situations and therefore the same gain values. Since there is no distinction between cells with the same gain values but belonging to different clusters, the FM algorithm may well start to work on many clusters simultaneously, trying to pull them out of unfavorable situations. However, cell movement is a two-way process, and while some cells in a cluster are moved from V_2 to V_1, other cells in the same cluster might be moved from V_1 to V_2. Thus the cluster can be locked in the cutset at an early stage —a cluster is said to be *locked* in the cutset if it has locked cells in both subsets of the partition. This is the situation of clusters C_1 and C_2 in Figure 1(b). Unfortunately, the moves made at an early stage (before the maximum partial sum point) are the actual moves. Hence these clusters will not be pulled out from the cutset in the current pass, and in later passes the same scenario may reappear.

Consider, for example, the simple graph shown in Figure 2(a) that shows a two-level clustered structure. Cells u_1^{11}, u_2^{11}, u_3^{11} and u_4^{11} form a strongly connected subcluster C_1^1; other subclusters C_1^2, C_2^1 and C_2^2 are similarly formed. Subclusters C_1^1 and C_1^2 construct a higher level but less densely interconnected cluster C_1. Similar is the case for C_2, which is composed of sublusters C_2^1 and C_2^2. After a random partition, all the clusters as well as the subclusters straddle the cutline. Initial FM gain calculation gives the gain values indicated beside each cell in the figure. Cells u_3^{11} and u_3^{22} belong to different clusters, but have similar situations, and hence the same gain of 4. We assume a 50%-50% balance criterion in which cells move alternately between the two subsets V_1 and V_2. The first four moves are shown by the numbered dashed arrows in Figure 2(a). FM quickly reaches the local minimum of cutsize 4 (further moves will be reversed finally since the current point is the maximum partial sum point). While FM succeeded in moving out subclusters, it locked the

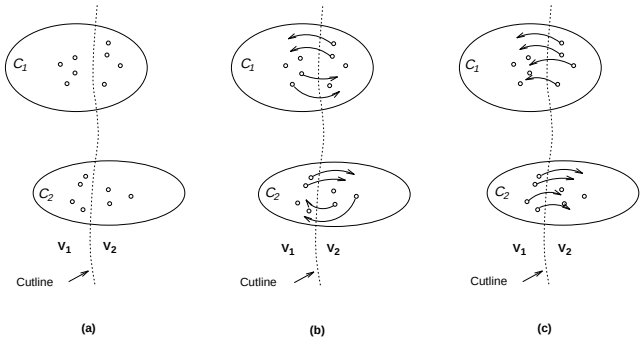

Figure 1: (a) In the initial partition, clusters straddle the cutline as a result of random cell assignment. (b) FM locks clusters on the cutline by moving cells within one cluster in both directions. (c) Better approaches pull clusters out from the cutline by moving cells within one cluster in a single direction.

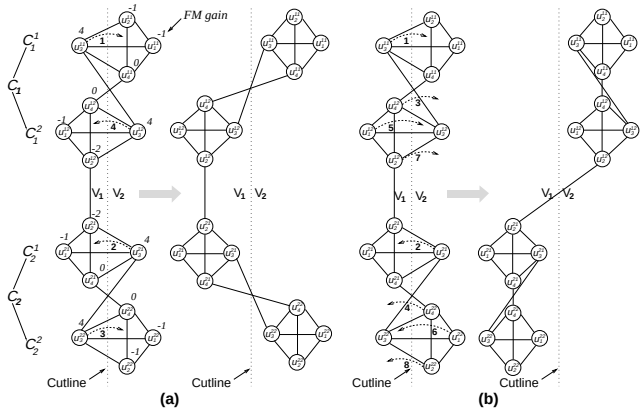

Figure 2: (a) FM only pulls out subclusters and finds a local minimum in cutsize. (b) The new approach pulls out clusters and finds the optimum cut.

higher level clusters C_1 and C_2 on the cutline. Therefore it missed the optimal cut of one that can be easily identified in the figure.

A mechanism is thus needed to aid iterative improvement algorithms in pulling out clusters from the cutset. We propose a cluster-oriented framework for gain calculation and base-cell selection that focuses on nets connected to moved cells. It can be overlaid on any iterative-improvement algorithm with any cell-gain calculation scheme. It implicitly promotes the move of an entire cluster by dynamically assigning higher weights to nets connected to recently moved cells. This greatly enhances the probability of finding a close-to-optimum cut in a circuit. We also propose an extended version of this algorithm that tries to identify clusters explicitly and then move them out from the cutset.

3.2. Considering Clusters in Iterative Improvement Methods

We first re-examine the cell gain calculation of FM. Initially, cell gains are calculated based on the immediate benefits of moving cells. After a cell is moved, the gains of its neighbors are updated. At any stage in the move process, the *total* gain of a cell can be broken down as the sum of the initial gain component and the updated gain component. The total gain indicates the overall situation of a cell, while the updated gain component reflects the change in the cell's status due to the movements of its neighbors.

An intuitive solution to the problem of an iterative-improvement scheme "jumping around" and working on different clusters simultaneously, as illustrated in Figs. 1(b) and 2(a), thus locking them in the cutset, is to make cell movement decisions based primarily on their updated gain components. This minimizes distractions during the cluster-pulling effort caused by cells not in the cluster currently being moved, but with high total gains. In other words, it allows the algorithm to concentrate on a single cluster at a time for moves in one direction; note that the updated gain component of a cell reflects its goodness for moving with regard to the cluster currently being pulled from the cutset. The initial gain of a cell, however, provides useful information for choosing the starting seed for removal of a cutset-straddling cluster—the cell with the highest gain is most likely in such a cluster, and thus a very good starting point. Once the move process has begun, nets connected to moved cells should be given more weights so that the updated gain components of cells become more important than their initial gains. The utility of giving more weight to nets connected to moved cells (and hence to the updated gain components of cells) in facilitating the movement of clusters from the cutset is established in the following set of results proved in [17].

We consider an iterative improvement partitioning process like FM and assume that the probability (f_{int}) that an edge connects a pair of cells inside the cluster is uniformly distributed, the probability (f_{ext}) that an edge connects a cell in C to a cell outside C is also uniformly distributed, and $f_{int} > f_{ext}$. This is similar to the *uniformly distributed random graph* model used by Wei and Cheng in [6].

Theorem 1 *If a cluster C is divided by the cutline into subsets $C_1 \in V_1$ and $C_2 \in V_2$, and C_1 is moved to V_2 by a sequence of moves of its cells, then the cutsize of the partition will decrease, if initially $|C_1| \leq |C_2|$, or first increase and then decrease, if initially $|C_1| > |C_2|$.*

Figure 4 illustrates Theorem 1. This theorem has also been empirically validated; see [17].

Assume that originally all net weights are one and when updating the gain of a cell, each net connected to moved cells is assigned a weight of at least $2p_{max}$, where recall that p_{max} is the maximum cell degree in the circuit.

Observation 1 *Once a cluster starts to move from V_1 to V_2, there is a high probability that the whole cluster will be removed from the cutset.*

The example of Fig. 2 can be used to demonstrate the advantage of the above approach. The cell gains are first computed and the base cell is again u_3^{11}. The first two moves of u_3^{11} and u_3^{21} bring large negative gains to cells u_3^{12} and u_3^{22} through the weighted edges. Therefore in the subsequent two moves, they are not selected as would be the case in FM. Instead, u_4^{12} becomes the top cell in V_1, and is selected as the next cell to move. The sequence of the first few moves are indicated by numbered arrows in the figure (for details see [17]). After eight moves, C_1 and C_2 are moved out from the cutset, and we obtain the optimal cut of one. Thus this process escapes the local minimum of four in which the original FM algorithm was trapped.

3.3. A Cluster-Oriented Iterative-Improvement Partitioner

From the above discussion, we propose a general gain calculation and base-cell selection framework CLIP (CLuster-oriented Iterative-improvement Partitioner), presented in Figure 3, that can be applied to any FM-type iterative improvement algorithm. For implementation convenience, we set the

Algorithm `CLIP`

1. Calculate the initial gain of all cells according to the iterative improvement algorithm of choice (e.g., FM, LA, PROP[16]).
2. Insert the cells into some sorted data structures (free-cell store) T_1 and T_2 for subsets V_1 and V_2, respectively. Select the maximum gain cell $u \in V$ as the first base cell to move.
3. Clear the gain of all cells to zero while maintaining their original ordering in the data structure.
4. Move u and update the gain of its neighbors and their ranks in the data structure as done in the chosen iterative improvement algorithm. The gain of a cell now only contains the updated part.
5. Choose the base cell based on the cell's updated gain and the balance criterion. Move the cell, update its neighbors.
6. **Repeat** Step 5 until all cells are moved.
7. Find the point in the move sequence which corresponds to the minimum cutsize, and reverse all the moves after this point.

Figure 3: One pass of CLIP (CLuster-oriented Iterative-improvement Partitioner).

cell gains to zero after the initial gain calculation, but order them according to their initial gains. Cell gains are updated as in the original algorithm over which CLIP is overlaid. Zeroing of initial gains followed by gain updating is equivalent to giving nets connected to moved cells a weight of infinity over nets with no moved cells.

After the first pass, most strongly connected clusters will probably have been removed from the cutset. The few clusters left in the cutset can be removed in the subsequent passes. In later passes, another advantage of the above cluster-oriented scheme is that clusters lying entirely in one subset can be easily moved to the other subset. This is because cell gains being cleared to zero in the initial stage causes cells in a cluster to have less inertia in staying inside their original subset. The benefit of cluster movement between subsets is that larger but less densely connected clusters (*superclusters*) can be removed from the cutset by moving their densely-connected constituent clusters from one subset to the other. By rearranging clusters between the two subsets in this way, subsets V_1 and V_2 of the final partition will become the two largest but most weakly connected superclusters, which implies that the cutsize will be small. As opposed to FM, which tends to do only local improvement within large clusters, the above new scheme can explore a wider solution space, and hence has less dependence on the initial partition.

Compared to other clustering-based approaches, such as bottom-up compaction [18], top-down clustering [8] and vertex ordering [12], CLIP does not explicitly bind cells together as inseparable clusters. Instead, cells can be implicitly regrouped into different clusters in subsequent passes. Both the moving and possible regrouping of clusters are guided directly by the ultimate objective—cutsize reduction. An advantage of this approach is that CLIP has more freedom to search for the optimum cut.

3.4. A Cluster Detection Method

Although the new framework CLIP has significant advantages over the traditional iterative improvement approach, it is possible to do even better. We start by asking the following questions. First, how do we know when a cluster has been pulled out? Furthermore, once we finish pulling out the first cluster, how do we select the next starting point?

We address the former question first by determining what happens when a cutline sweeps across a cluster—this is equivalent to a cluster being pulled out from the cutset. From Theorem 1, as we move a cluster from the cutset, the overall cutsize will decrease until the cluster is entirely removed from the cutset. In a practical partitioner, some external cells

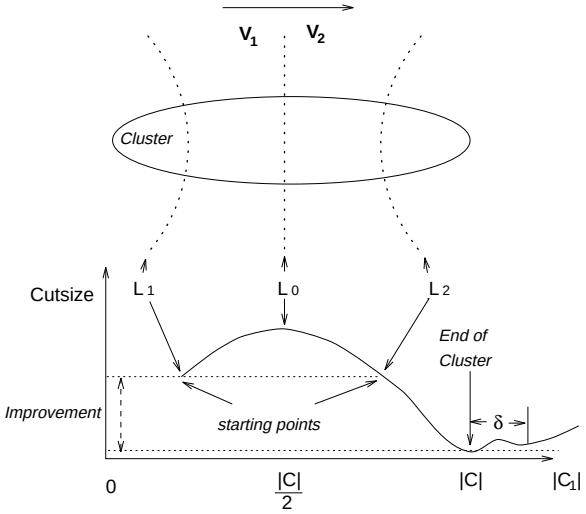

Figure 4: The cutsize change with the move of a cluster as indicated in Theorem 1. When the cutsize does not improve, it indicates the end of a cluster.

Test Case	# of Cells	# of Nets	# of Pins	Test Case	# of Cells	# of Nets	# of Pins
s1423	619	538	1528	p2	3014	3029	11219
sioo	664	408	1882	s9234	5866	5844	14065
s1488	686	667	2079	biomed	6514	5742	21040
balu	801	735	2697	s13207	8772	8651	20606
p1	833	902	2908	s15850	10470	10383	24712
bm1	882	903	2910	industry2	12637	13419	48404
t4	1515	1658	5975	industry3	15406	21924	68290
t3	1607	1618	5807	s35932	18148	17828	48145
t2	1663	1720	6134	s38584	20995	20717	55203
t6	1752	1541	6638	avq.small	21918	22124	76231
struct	1952	1920	5471	s38417	23949	23843	57613
t5	2595	2750	10076	avq.large	25178	25384	82751
19ks	2844	3282	10547				

Table 1: Benchmark circuit characteristics.

may be moved across the cutline due to their connections to moved cells in the cluster. However, since they are randomly distributed across many clusters (and thus do not belong to a specific cluster), their contribution to the overall cutsize change will not be significant. As a result, we obtain a cutsize change similar to that illustrated in Figure 4, which is a pictorial depiction of Theorem 1. Referring to this figure, the movement of a cluster starts from either point L_1 or L_2. After it is removed from the cutset, there is an overall improvement in cutsize. If in subsequent moves no other cluster starts getting pulled out from the cutset, the cutsize will not improve further.

From this reasoning we propose the following cluster detecting criterion: *After the move process reaches a positive maximum improvement point, and there is no further improvement in the following δ moves, we say that a cluster has been moved out at the maximum improvement point; δ is a parameter of the algorithm.* Referring to the cutsize curve in Figure 4, the δ cells moved after the minimum cutsize is reached do not belong to the previous cluster. This is in contrast to the two cluster detection criteria that Saab proposed in his compaction algorithm [18], viz., (1) The cutsize decreases for the first time after a sequence of moves; (2) The last moved cell in the sequence has a positive gain. We derived our detection criterion from the analytical results in Section 3.2., and we use the partial sum of gains of the sequence of node moves in our criterion, instead of individual cell gains as in [18].

We now address the second question raised at the beginning of this subsection. After reversing the δ moves, we come back to the end point of the current cluster. Subsequently, we need to select the next seed to start the move of another cluster. For this purpose, the updated gain components of cells should not be the determining factor. Rather, the total gains of cells, which reflect their overall situation, is more useful. Similar to the situation at the beginning of the pass, the cell with the maximum total gain is a good seed to start with. Following this, however, unlike at the beginning of the pass, gain components of nodes are selectively zeroed. Negative components due to connections to locked nodes are retained, since they signify attachment to a moved cluster, and a node with a significant negative gain most probably belongs to such a cluster and should not be moved; see Fig. 5 for more details.

From the above discussion, a more sophisticated algorithm CDIP (Cluster-Detecting Iterative-improvement Partitioner) is presented in Fig. 5. This is an extension of CLIP and can also be applied to any FM-type iterative improvement algorithm. Note that in both CLIP and CDIP, the move process removes (disjoint) clusters from the cutset to both V_1 and V_2 in an interleaved manner. Though in CDIP there can be intermediate configurations in which the balance criterion is relaxed (this is not allowed in CLIP, since it is not crucial to its operation), care is taken that it is ultimately satisfied; see Fig. 5.

By using either of the two new algorithms, CLIP or CDIP, we can find a good cut through weakly connected clusters. In order to obtain even better results, we can apply the original iterative improvement algorithm as a post-processing phase to fine-tune the partition.

3.5. Complexity Analysis

In an implementation, both CLIP and CDIP have to be overlaid on some chosen iterative improvement algorithm. Let n be the number of cells, e the number of nets, p the number of pins and d the average number of neighbors of a cell. It is easy to show that CLIP does not increase the order of time complexity of the original algorithm (which will always

Algorithm `CDIP`

1. Calculate the initial gain of all cells according to the iterative improvement algorithm of choice.
2. Insert the cells into sorted data structures T_1 and T_2 for subsets V_1 and V_2, respectively. Select the maximum gain cell $u \in V$ as the first base cell to move.
3. Clear the gain of all cells to zero while keeping their original ordering in T_i ($i = 1, 2$).
4. Move u and update the gain of its neighbors and their ranks in T_i ($i = 1, 2$) as done in the chosen iterative improvement algorithm. Start to count the move index j and to calculate the partial sum $S_j^{i,1}$ for this first cluster, where $u \in V_i$. The gain of a cell now only contains the updated part.
5. **Repeat** Steps 6 to 7 until all cells are moved and locked.
6. Choose the base cell based on the cell gain, and the balance criterion with its allowable intermediate relaxation. Move the cell, and update the neighbors as done before.
7. **If** the current maximum partial sum $S_p^{i,k} > 0$, and the current move index $q \geq p + \delta$, **then**
   (a) Reverse the moves from q to $p + 1$.
   (b) Choose the free cell $v \in V_i$ with the maximum *total* gain as the next base cell.
   (c) For each free cell in V_i, clear the cell gain except the gain component from nets connected to the locked cells in the same subset V_i. Reorder cells in T_i according to the modified gain.
   (d) Move the base cell v, update neighbors and start the count of new move index j and the calculation of new partial sum $S_j^{i,k+1}$ from this move.
8. Find the maximum prefix point in the moving sequence that satisfies the balance criterion (there may be intermediate relaxations at some points), and reverse all the moves after this point.

Figure 5: One pass of CDIP (Cluster-Detecting Iterative-improvement Partitioner).

Test Case	Minimum of 20 runs								Average of 20 runs							
	Cut Size					Improvement %			Cut Size					Improvement %		
	FM	LA3	CLIP -FM	CLIP -LA3	CDIP -LA3	CLIP -FM	CLIP -LA3	CDIP -LA3	FM	LA3	CLIP -FM	CLIP -LA3	CDIP -LA3	CLIP -FM	CLIP -LA3	CDIP -LA3
s1423	17	16	15	16	15	11.8	5.9	11.8	24.2	22.3	21.8	19.6	20.6	9.9	18.8	14.7
sioo	31	25	37	25	25	-16.2	19.4	19.4	47.5	25.4	49.9	25.3	25.1	-4.8	46.7	47.1
s1488	48	42	43	42	41	10.4	12.5	14.6	53.5	49.1	46.9	45.5	46.4	12.3	14.8	13.2
balu	27	27	27	27	27	0.0	0.0	0.0	38.1	36.1	38.9	32.5	33.9	-2.1	14.6	11.2
p1	47	52	52	52	52	-9.6	-9.6	-9.6	74.9	68.5	65.8	61.8	61.5	12.1	17.6	17.9
bm1	54	53	49	52	52	9.3	3.7	3.7	79.5	67.5	65.0	60.4	59.8	18.3	24.1	24.8
t4	87	82	56	51	52	35.6	41.4	40.2	129.3	117.2	77.0	72.9	71.7	40.4	43.6	44.6
t3	75	80	57	57	57	24.0	24.0	24.0	106.8	106.2	72.3	71.6	66.8	32.2	32.9	37.4
t2	149	126	89	92	90	40.3	38.3	39.6	182.1	148.1	105.2	106.8	101.0	42.3	41.4	44.5
t6	67	70	60	60	60	10.4	10.4	10.4	94.2	84.4	70.1	71.8	70.0	25.5	23.7	25.6
struct	46	44	37	33	36	19.6	28.3	21.7	58.0	49.6	45.6	46.8	46.8	21.3	19.3	19.4
t5	127	99	75	80	74	40.9	37.0	41.7	183.6	165.0	89.0	90.7	89.8	51.5	50.6	51.1
19ks	140	130	119	107	105	15.0	23.6	25.0	171.7	169.0	150.3	136.1	134.6	12.4	20.7	21.6
p2	212	149	149	142	152	29.7	33.0	28.3	273.9	233.4	233.2	208.8	193.2	14.8	23.8	29.5
s9234	59	43	49	47	44	16.9	20.3	25.4	84.7	81.1	89.5	80.8	77.2	-5.4	4.5	8.9
biomed	83	90	84	84	83	-1.2	-1.2	0.0	117.4	170.7	108.4	102.1	102.2	7.7	13.0	13.0
s13207	98	85	98	68	70	0.0	30.6	28.6	122.6	118.9	123.5	100.5	93.9	-0.8	18.0	23.4
s15850	109	87	80	73	67	26.6	33.0	38.5	176.9	140.4	140.9	106.7	107.0	20.3	39.7	39.5
industry2	264	422	260	205	190	1.5	22.3	28.0	627.5	732.4	369.6	332.8	293.6	41.1	47.0	53.2
Subtotal	1740	1722	1436	1313	1292	17.5	24.5	25.7	2646	2585	1963	1773	1694	25.8	33.0	36.0
industry3	272	504	261	261	243	4.0	4.0	10.7	506.0	758.0	376.6	421.6	358.6	25.6	16.7	29.1
s35932	85	168	102	73	73	-16.7	14.1	14.1	210.3	231.4	127.1	79.0	83.0	39.6	62.4	60.5
s38584	100	85	49	55	47	51.0	45.0	53.0	299.8	271.2	83.2	103.3	95.7	72.2	65.5	68.1
avq.small	347	608	223	146	148	35.7	57.9	57.3	578.6	815.5	335.1	309.9	311.7	42.1	46.5	46.1
s38417	240	284	78	73	79	67.5	69.6	67.1	384.1	408.2	136.7	114.0	108.7	64.4	70.3	71.7
avq.large	350	398	216	138	145	38.3	60.6	58.6	755.0	693.4	305.2	349.1	280.8	59.6	53.8	62.8
Subtotal	1394	2047	929	746	735	33.4	46.5	47.3	2734	3178	1363	1377	1238	49.1	48.6	53.8
Total	3134	3769	2365	2059	2027	24.5	34.3	35.3	5380	5763	1963	1773	1694	38.2	41.4	45.5
Average of % improvement						17.8	25.0	26.1						26.1	33.2	35.2

Table 2: Comparisons of CLIP and CDIP (applied to FM and LA3) to FM. CLIP-LA3 results are for $\delta = 50$. Subtotals shown correspond first to medium-size circuits and then to large circuits.

be $\Omega(p)$). CLIP-FM and CLIP-LA (CLIP applied to FM and LA, resp.) can thus be implemented with $\Theta(p)$ complexity if a bucket list structure is used.

CDIP has two major additional operations beyond those in the CLIP algorithm. After each detection of a cluster: (1) The δ moves are reversed, and (2) The free cells are reordered. Assuming c clusters are detected in a pass, the first operation implies $c\delta$ additional moves over the entire pass, and the second operation causes c reorderings of all free cells. For a bucket list structure, where the insertion of a cell into a bucket is a constant time operation, the complexity of the first operation is $O(c\delta d)$ (each extra move requires $O(d)$ time for updating d neighbors and reinserting them in the bucket data structure), and the complexity of the second operation is $O(cn)$ over the entire pass. Thus the complexity of CDIP is $O(max(c\delta d, cn, p))$ for CDIP-FM and CDIP-LA for one pass. Empirical results presented later show that CDIP is quite fast.

4. Experimental Results

All experiments have been done on ACM/SIGDA benchmark circuits whose characteristics are listed in Table 1. The circuit netlists were acquired from the authors of [12] and [13]. All cutset results are for the 45-55% balance criterion.

4.1. Comparisons to FM

Table 2 presents the results of applying CLIP to FM and LA3 (LA algorithm with lookahead level of 3). Both the minimum and average cutsizes over 20 runs are greatly improved compared to their corresponding original schemes. The overall minimum-cutsize improvements are 24.5% for CLIP-FM over FM and 45.4% for CLIP-LA3 over LA3. Note also from the table that while LA3 performs slightly better than FM for small to medium-size circuits, it performs much worse for large circuits—for an explanation of this phenomenon the interested reader is referred to [17]. However, CLIP-LA3 now performs much better than FM (by 24.5% for medium-size benchmarks and 46.5% for large-size benchmarks, for an overall improvement of 34.3%). As is clearly evident, the most improvements of the new schemes over FM are obtained for large circuits. The largest improvement of CLIP-LA3 over FM is about 70% for the circuit *s38417*. This clearly demonstrates the new clustering-based schemes' ability to tackle large circuits. The cluster detection method CDIP-LA3, obtained by overlaying CDIP on LA3, performs even better. The minimum and average cutsizes are improved over those of FM by 35.3% and 45.5%, respectively. Both the minimum and average cutsizes are also superior to those of CLIP-LA3. This indicates that CDIP is a better and more stable partitioner than CLIP.

4.2. Comparisons to Paraboli and MELO

Finally, in Tables 3 and 4, we compare the original iterative improvement and the new cluster-oriented iterative improvement algorithms to two state-of-the-art partitioning methods, the placement-based algorithm Paraboli [13] and the spectral partitioner MELO [14]. Here, all the new clustering-based iterative improvement algorithms are further improved by the corresponding original schemes as indicated at the end of Section 3.4. (e.g., CLIP-FM$_f$ is CLIP-FM followed by FM improvement). Since FM is very fast, we perform 100 runs of both FM and CLIP-FM$_f$; all other iterative improvement algorithms' results are for 20 runs.

First, it is clear from the tables that the original FM algorithm can obtain good results for medium-size circuits. For this set of benchmarks, it is about 4% better than MELO, and 1% better than Paraboli in total cut. However, for large size circuits, it falls far behind Paraboli (by -22.5% in total cut). This confirms our earlier discussion on the shortcomings of previous iterative methods. After using the clustering-based techniques, CLIP and CDIP, all of the four new algorithms CLIP-FM$_f$, CLIP-LA3$_f$, CLIP-PROP$_f$ and CDIP-LA3$_f$, are

Test Case	Cut Size						% Improvement over Paraboli				
	Paraboli	FM	CLIP -FM$_f$	CLIP -LA3$_f$	CDIP -LA3$_f$	CLIP -PROP$_f$	FM	CLIP -FM$_f$	CLIP -LA3$_f$	CDIP -LA3$_f$	CLIP -PROP$_f$
s1423	16	17	15	15	15	15	-5.9	6.2	6.2	6.2	6.2
sioo	45	25	25	25	25	25	44.4	44.4	44.4	44.4	44.4
s1488	50	46	43	42	41	43	8.0	14.0	16.0	18.0	14.0
balu	41	27	27	27	27	27	34.1	34.1	34.1	34.1	34.1
p1	53	47	47	51	47	51	11.3	11.3	3.8	11.3	3.8
struct	40	41	33	33	36	33	-2.4	17.5	17.5	10.0	17.5
p2	146	182	148	142	151	152	-19.8	-1.4	2.7	-3.3	-3.9
s9234	74	51	44	45	44	42	31.1	40.5	39.2	40.5	43.2
biomed	135	83	83	83	83	84	38.5	38.5	38.5	38.5	37.8
s13207	91	78	76	66	69	71	14.3	16.5	27.5	24.2	22.0
s15850	91	104	75	71	59	56	-12.5	17.6	22.0	35.2	38.5
industry2	193	264	174	200	182	192	-26.9	9.8	-3.5	5.7	0.5
Subtotal	975	965	790	800	779	791	1.0	19.0	17.9	20.1	18.9
industry3	267	263	241	260	243	243	1.5	9.7	2.6	9.0	9.0
s35932	62	85	83	73	73	42	-27.1	-25.3	-15.1	-15.1	32.3
s38584	55	63	47	50	47	51	-12.7	14.5	9.1	14.5	7.3
avq.small	224	297	200	129	139	144	-24.6	10.7	42.4	37.9	35.7
s38417	49	147	66	70	74	65	-66.7	-25.8	-30.0	-33.8	-24.6
avq.large	139	350	185	127	137	143	-60.3	-24.9	8.6	1.4	-2.8
Subtotal	796	1205	822	709	713	688	-33.9	-3.2	10.9	10.4	13.6
Total cut	1771	2170	1612	1509	1492	1479	-22.5	8.8	14.8	15.8	17.5
Average of per-ckt % improvements							-4.2	11.6	14.8	15.5	16.5

Table 3: Comparisons of various iterative improvement algorithms to Paraboli [13]. The results for the clustering-based iterative-improvement algorithms in the table have been further improved by their corresponding original schemes (indicated by the subscript $_f$). FM and CLIP-FM$_f$ results are the best cutsizes from 100 runs, while results of our other algorithms are the best of 20 runs. CDIP-LA3$_f$ results are for $\delta = 50$. Subtotals shown correspond first to medium-size circuits and then to large circuits.

able to obtain cutsizes that are overall better than Paraboli's. Total cut improvements range from 8.8% to 17.5%.

The best results are obtained by applying CLIP to PROP (CLIP-PROP). PROP is a probability-based iterative improvement partitioner [16] that calculates cell gains based on the probabilities of nets being actually moved in a pass. Thus the cell gain calculation is more accurate than that of either FM or LA. Hence CLIP-PROP overcomes the two fundamental shortcomings in traditional iterative improvement methods—inaccurate gain calculation and lack of a global view of the cluster-oriented structure of circuits. It thus emerges as a very powerful partitioning tool and performs about 17% better than Paraboli. When compared to MELO [14], for which only results on medium-size circuits are given, all the new algorithms show about 23% better results in total cutsize.

4.3. Run Time Comparisons

The run times of the iterative improvement algorithms are very favorable compared to other purely clustering-based algorithms like Paraboli and MELO. Run times of Paraboli and MELO are reported for the DEC3000 Model 500 AXP and SUN SPARC10 workstations, respectively, while we have executed all CLIP and CDIP based algorithms, as well as FM and LA, on the SUN SPARC5 Model 85 workstation; see [17] for a detailed table. The data structures used to store free cells for FM and LA are bucket structures as proposed in [3] and [4], respectively. Despite the $O(max(c\delta d, cn, p))$ worst-case time complexity of CDIP-LA3$_f$, in practice it uses less than twice the CPU time of CLIP-LA3$_f$, which has a linear run time. PROP uses a tree structure which makes it easy to accommodate arbitrary net weight as is required in performance-driven partitioning [7, 5]. Yet the run time of CLIP-PROP$_f$ is quite reasonable. The total CPU times of all four new algorithms are less than that of Paraboli. Assuming the same speed for the three different workstations, CLIP-FM$_f$ is 1.9 times, CLIP-LA3$_f$ is 3.3 times, and both CDIP-LA3$_f$ and CLIP-PROP$_f$ are 1.8 times faster than Paraboli. CLIP-PROP$_f$ is comparable to MELO in run time, while CLIP-FM$_f$, CLIP-LA3$_f$ and CDIP-LA3$_f$ are faster than MELO by factors of 1.2, 1.7 and 1.2, respectively. This also means that if equal CPU times are allocated, the new algorithms can perform more runs and generate even better cutsizes than those presented in Tables 3 and 4.

5. Conclusions

We proposed a new clustering-based approach that greatly enhances the performance of iterative improvement methods. The new approach incorporates clustering mechanisms naturally into traditional FM-type algorithms. This revives the power of fast move-based iterative improvement algorithms, making them capable of dealing with large-size circuits. They are significantly better, in terms of both cutset quality and speed, than current state-of-the-art algorithms like Paraboli [13] and MELO [14] that are purely clustering-based, and deserve new attention in the development of VLSI CAD tools.

Acknowledgements

This work was supported partly by NSF grant MIP-9210049. We thank the anonymous referees for their useful comments.

References

[1] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell System Tech. Journal*, vol. 49, Feb. 1970, pp. 291-307.

[2] D. G. Schweikert and B. W. Kernighan, "A Proper Model for the Partitioning of Electrical Circuits", *Proc. 9th Design automation workshop*, 1972, pp. 57-62.

[3] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions", *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175-181.

[4] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks", *IEEE Trans. on Comput.*, vol. C-33, May 1984, pp. 438-446.

[5] M. Marek-Sadowska and S.P. Lin, "Timing driven placement", *Proc. IEEE/ACM International Conference on CAD*, 1989, pp. 94-97.

[6] Y.C. Wei and C.K. Cheng, "Towards efficient hierarchical designs by ratio cut partitioning", *Proc. Int'l. Conf. Computer-Aided Design*, 1989, pp. 298-301.

Test Case	Cut Size						% Improvement over MELO				
	MELO	FM	CLIP -FM$_f$	CLIP -LA3$_f$	CDIP -LA3$_f$	CLIP -PROP$_f$	FM	CLIP -FM$_f$	CLIP -LA3$_f$	CDIP -LA3$_f$	CLIP -PROP$_f$
balu	28	27	27	27	27	27	3.6	3.6	3.6	3.6	3.6
p1	64	47	47	51	47	51	26.6	26.6	20.3	26.6	20.3
bm1	48	49	47	51	47	47	-2.0	2.1	-5.9	2.1	2.1
t4	61	80	53	49	48	52	-23.8	13.1	19.7	21.3	14.8
t3	60	62	56	56	57	57	-3.2	6.7	6.7	5.0	5.0
t2	106	124	87	92	89	87	-14.5	17.9	13.2	16.0	17.9
t6	90	60	60	60	60	60	33.3	33.3	33.3	33.3	33.3
struct	38	41	33	33	36	33	-7.3	13.2	13.2	5.3	13.2
t5	102	104	74	80	74	77	-1.9	27.5	21.6	27.5	24.5
19ks	119	130	109	104	104	104	-8.5	8.4	12.6	12.6	12.6
p2	169	182	148	142	151	152	-7.1	12.4	16.0	10.7	10.1
s9234	79	51	44	45	44	42	35.4	44.3	43.0	44.3	46.8
biomed	115	83	83	83	83	84	27.8	27.8	27.8	27.8	27.0
s13207	104	78	76	66	69	71	25.0	26.9	36.5	33.7	31.7
s15850	52	104	75	71	59	56	-50.0	-30.7	-26.8	-11.9	-7.1
industry2	319	264	174	200	182	192	17.2	45.5	37.3	42.9	39.8
Total cut	1554	1486	1193	1210	1177	1192	4.4	23.2	22.1	24.3	23.3
Average of per-ckt % improvements							3.2	17.4	17.0	18.8	18.5

Table 4: Comparisons of various iterative improvement algorithms to MELO [14]. The results in the table have been further improved by the original schemes (indicated by the subscript $_f$). FM and CLIP-FM$_f$ results are the best cutsizes from 100 runs, while results of our other algorithms are the best of 20 runs. CDIP-LA3$_f$ results are for $\delta = 50$.

[7] M.A.B. Jackson, A. Srinivasan and E.S. Kuh, "A fast algorithm for performance driven placement", *Proc. IEEE/ACM International Conference on CAD*, 1990, pp. 328-331.

[8] Y. C. Wei and C. K. Cheng, "An Improved Two-way Partitioning Algorithm with Stable Performance", *IEEE Trans. on Computer-Aided Design*, 1990, pp. 1502-1511.

[9] L. Hagen and A. B. Kahng, "Fast Spectral Methods for Ratio Cut Partitioning and Clustering", *Proc. IEEE Intl. Conf. Computer-Aided Design*, 1991, pp. 10-13.

[10] J. Cong and M. Smith, "A bottom-up clustering algorithm with applications to circuit partitioning in VLSI designs", *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp. 755-760.

[11] S. Dutt, "New faster Kernighan-Lin-type graph-partitioning algorithms", *Proc. IEEE/ACM International Conference on CAD*, Nov. 1993.

[12] C. J. Alpert and A. B. Kahng, "A General Framework for Vertex Orderings, With Applications to Netlist Clustering", *Proc. IEEE Intl. Conf. Computer-Aided Design*, 1994, pp. 63-67.

[13] B. M. Riess, K. Doll and F. M. Johannes, "Partitioning Very Large Circuits Using Analytical Placement Techniques", *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp. 646-651.

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

[15] L.W. Hagen, D. J.-H. Hwang and A.B. Kahng, "On implementation choices for iterative improvement partitioning methods", *Proc. European Design Automation Conf.*, Sept. 1995, pp. 144-149.

[16] S. Dutt and W. Deng, "A Probability-Based Approach to VLSI Circuit Partitioning", *Proc. ACM/IEEE Design Automation Conf.*, June 1996, pp. 100-105. (**Recepient of the best-paper award**.)

[17] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques", Technical Report, Department of Electrical Engr., University of Minnesota, Nov. 1995.
This report is available at ftp site ftp-mount.ee.umn.edu in file pappdw96-ext.ps in directory /pub/faculty/dutt/vlsi-cad/papers.

[18] Y. G. Saab, "A Fast and Robust Network Bisection Algorithm", *IEEE Trans. Computers*, 1995, pp. 903-913.