PARTITIONING BY ITERATIVE DELETION

Patrick H. Madden [*]

State University of New York at Binghamton
pmadden@cs.binghamton.edu

ABSTRACT

Netlist partitioning is an important and well studied problem. In this paper, a linear time partitioning approach based on *iterative deletion* is presented. We use the partitioning problem to allow a fair comparison of the iterative deletion approach with well known iterative improvement methods. For partitioning problems with a range of edge weights, and for multi-way partitioning, the iterative deletion approach can outperform the iterative improvement method. The algorithmic approach is flexible and can support complex cost functions directly.

1. INTRODUCTION

The netlist partitioning problem is well known; [3] provides a recent survey of work on this problem. The common formulation involves a hypergraph $H(V, E)$, with n vertices $V = \{v_1, v_2, ...v_n\}$. Each edge $e_i \in E$ of the hypergraph connects a subset of the vertices. If we *partition* the hypergraph, we determine a group of non-intersecting subsets of V. Each group is known as a *cluster*, and when we wish to obtain a pair of clusters, this is a *bipartition*. Partitioning into k clusters $c_1, c_2, ...c_k$ is known as *multi-way* or $k - way$ partitioning. When an edge e connects vertices in more than one cluster, we say that this edge is *cut*. A common objective is to minimize the total number (or total weight) of cut edges, while meeting constraints on the total number (or total weight) of vertices in each cluster.

A number of different algorithmic approaches have been considered for the netlist partitioning problem. In this paper, we investigate the relatively uncommon *iterative deletion* approach. We compare performance of this approach to more widely known iterative improvement approaches, and identify areas in which iterative deletion outperforms iterative improvement.

The remainder of this paper is organized as follows. In Section 2, we briefly describe previous partitioning algorithms, with a focus on iterative improvement move-based approaches. In Section 3, we describe a number of issues related to the use of partitioning for circuit placement applications. Section 4 describes our iterative deletion based partitioning approach in detail. Experimental results are presented in Section 5, where we compare our approach to a number of well known partitioning algorithms, for both bipartitioning and multi-way partitioning problems. The paper concludes with some remarks in Section 6.

[*]This work is partially supported by the SUNY Binghamton Office of Research and Sponsored Programs.

2. PREVIOUS WORK

The survey of [3] provides four general classifications for current approaches to the traditional partitioning problem. *Move-based* approaches are common; most begin with a randomly chosen initial partition, and iteratively move vertices between clusters to optimize the number of edges cut. Geometric approaches are also common, and approach the problem by obtaining linear orderings of vertices through *spectral* methods. For small problems, the processing power available on current workstations allows *combinatorial* methods to be used, and *clustering* approaches find partitions by merging closely related vertices iteratively.

The move-based approaches frequently employ *iterative improvement*, in which an initial partition is optimized by repeatedly moving a vertex from one cluster to another, or by swapping pairs of vertices. The algorithm by Kernighan and Lin (KL)[14] pioneered this approach; the algorithm by Fiduccia and Mattheyses (FM)[10] improves upon this general direction, providing an extremely efficient linear-time technique. The FM algorithm forms the basis of many current partitioning methods.

These iterative improvement approaches generally operate in a series of *passes*. Starting from an initial random partitioning, vertices are moved from cluster to cluster, with each vertex being moved once in a pass. After all vertices have been moved, a move-based iterative improvement approach *rolls back* to the best intermediate solution, and begins another pass.

In general, an iterative improvement approach such as FM can produce an extremely wide range of solution qualities, depending on the initial random partitioning. In order to obtain good quality solutions, it is common practice to run these algorithms repeatedly using different random seeds.

The iterative improvement based FM algorithm is also quite sensitive to the methods used to break ties during optimization; [11] observed that a Last-In-First-Out (LIFO) strategy resulted in a 43% improvement in minimum-cut bisection results, compared to a First-In-First-Out (FIFO) strategy.

The survey of [3] provides a description of the other major partitioning approaches.

2.1. Multilevel Partitioning

The current state of the art partitioning algorithm[13] employs a hybrid of clustering and iterative improvement. The “multilevel” approach clusters the initial hypergraph through a series of levels, each progressively “coarser” than the last. When the hypergraph reaches a specified granularity, traditional partitioning approaches can be applied. The partitioning solution at one level of coarseness can be used to provide a starting point for the next level of detail; rather than facing an initial random solution, each level begins with a high quality starting point. The substantial performance improvements of [8, 2, 13] indicate that multilevel formulations are extremely effective if minimized cut sizes are our final objective.

2.2. Multi-Way vs. Recursive Partitioning

If our objective is to obtain a k-way partition for values of k of 3 or greater, we can employ either a "flat" multi-way partitioning, or recursive bipartitioning. In experiments on large benchmarks with the multi-way partitioning algorithm K-FM[15], it was observed in [5] that recursive bipartitioning provided superior results. This is somewhat surprising, in that we may expect that the greater degree of freedom available through flat k-way partitioning would provide more opportunity for optimization. In practice, it appears that the flat multi-way iterative improvement approach encounters a large number of local minima which are difficult to escape; the tie-breaking schemes suggested in [11] provide little benefit, as few ties are encountered.

The partitioning approach of [5] improved performance through the *restriction* of moves considered. Rather than allowing a vertex to move from one cluster to any other, the clusters are "paired" at the beginning of each improvement pass, and all exchanges occur between the pairs. The reduction in considered moves actually improves overall performance, indicating that iterative improvement based approaches perform better when faced with fewer choices.

While recursive bipartitioning can be quite effective in minimizing cut sizes, solutions developed in this manner are less useful in circuit placement applications. With the recursive formulation, we lose the ability to accurately model inter-cluster routing costs. Additionally, the mapping of a recursive bipartition based k-way solution to a physical placement cannot be done directly. In Figure 1, we have an example in which the quality of the final placement depends on two different partitioning problems. If we have a connection between vertices (modules) v_i and v_j, the placement of one impacts the preferred location of the other; we have no way of knowing which partitioning is "best" from a global perspective.

3. PARTITIONING AND PLACEMENT CONSIDERATIONS

As partitioning algorithms are frequently used as a component of circuit placement approaches, we now focus on the relationship of partitioning solutions to placement problems. Circuit netlists can be mapped easily into the hypergraph formulation; in fact, many partitioning benchmarks are based on circuit netlists. A cell or module in a circuit design is equivalent to a vertex in the hypergraph, while a connecting signal net is equivalent to a hyperedge.

Module placements can be obtained through a series of partitionings; if we recursively divide the netlist into smaller and smaller clusters, a hierarchy with small numbers of inter-cluster connections can be obtained. By assigning each cluster to a physical region on the integrated circuit, we obtain a placement which has small numbers of inter-region connections. An early placement algorithm based on this approach is [4]. In [9], the KL algorithm was utilized to provide improved results. Recently, [12] applied an effective multilevel and multi-way partitioning algorithm, replacing the minimum cut objective with a more precise estimate of wiring length.

Figure 1 shows an application of bipartitioning to placement. We assume that we are to embed the hypergraph in a small rectangular region, and have an objective of minimizing the total length of the interconnecting edges. If we partition the initial netlist or hypergraph with vertices v_1, v_2, v_3, v_4 into two clusters containing v_1, v_2 and v_3, v_4, we can place them in two halves of our rectangular region. By recursively applying partitioning to the two clusters, we can determine exact locations for each vertex or circuit element. While we have minimized the number of edges cut

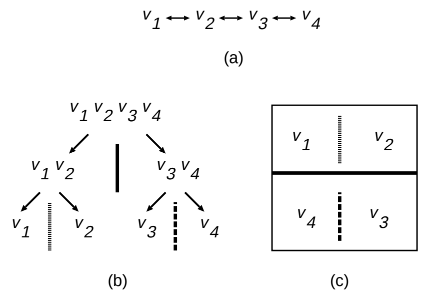

Figure 1. Recursive bipartitioning, as applied to a placement problem. In (a), we have an initial circuit or netlist. Hierarchical partitioning is applied in (b), and the result of this hierarchy can be mapped to the placement (c). In this figure, vertices that have been separated during partitioning may be adjacent in the placement. In general, a high quality partition is not equivalent to a high quality placement.

between clusters in this approach, we diverge from some objectives relevant to the circuit placement problem. This formulation does not address the relative adjacencies between regions in the final placement (in Figure 1, for example, v_1 is adjacent to v_4, although they are widely separated under the partitioning formulation). While we have relatively few inter-region connections, some of these connections may be excessively long.

Partitioning approaches play an active role in current circuit design, but recent advances in partitioning have not solved all the challenges of performance driven placement. Common partitioning objectives may minimize the number of connections between subcomponents, but fail to directly address concerns such as signal delay. A partitioning solution which reduces the total number of cut edges may do so at the expense of having one or more edges span a large number of clusters.

The impact of a handful of "long" interconnect wires on circuit performance is substantial. Due to the scaling of device dimensions, interconnect delay has increased. As delay increases quadratically with wire length, a solution which has a low number of long inter-region connections may be inferior to a solution with larger numbers of short inter-region connections. These few long wires dominate system delay, preventing high performance. [16] summarizes some of the interconnect constraints faced in current circuit designs.

VLSI integrated circuit netlists frequently have large numbers of vertices. Many designs have several hundred thousand logic elements[1], and these numbers are expected to grow. A partitioning algorithm which is effective for flat multi-way partitioning of large netlists is of interest for circuit placement applications. If we assign inter-cluster routing costs (to model physical wiring distances), and utilize a large number of clusters, we quickly obtain a rough circuit placement which may be a suitable starting point for local refinement.

The algorithm presented has been developed with a consideration of circuit placement objectives, but we address a traditional partitioning problem formulation here. In order to make a clear comparison of the iterative deletion and iterative improvement approaches, we do not employ any additional techniques. While the results of both iterative improvement and iterative deletion approaches can be enhanced greatly, these enhancements obscure the merits of the underlying approaches. For these reasons, our approach does not involve the following.

- *Clustering.* Clustering simplifies the hypergraph, and reduces the solution space considerably. To measure the relative performance of the two approaches, however, we are interested in each problem being "difficult."

If we cluster the partitioning problem, we can expect substantial reductions in cut size. While most connections in the placed result may be extremely short, a handful can be exceptionally long, resulting in high delay. Thus, clustering may be detrimental to solution quality for circuit placement applications. Clustering also obscures the delay of connections internal to the cluster, making performance driven design more difficult.

- *Recursive Bipartitioning.* By repeatedly bipartitioning a hypergraph, rather than attempting a "flat" partitioning, we may reduce the number of nets cut globally. We consider both bipartitioning and multi-way partitioning in order to evaluate the relative merits of the approaches under a range of conditions.

  As with clustering, we can expect connections which cross cut lines to have a wide range of physical lengths. It is in general difficult to predict which cluster a given vertex should be assigned to, and the hierarchical clusterings produced by recursive approaches may be difficult to map into circuit placements. For this reason, "flat" partitionings may be preferable.

4. PARTITIONING BY ITERATIVE DELETION

The method presented in this paper is based on the *iterative deletion* approach used in VLSI standard cell routing[7], and does not fall into the four general classifications described in [3].

The iterative deletion approach for standard cell global routing begins with a set of connections between module pins; while only a single segment may be required to obtain electrical connectivity, extra segments are included. Segments which are on cycles (and thus are not required for connectivity) are said to be *redundant*. In order to minimize total wire length and circuit area, redundant segments are iteratively removed in a greedy fashion, until a low wirelength and low area connected subset is obtained. This general approach has been continued in [6].

4.1. Partitioning Approach

While the partitioning and global routing problems may seem unrelated, the iterative deletion approach can be directly applied. We consider k-way partitioning into four clusters here, as it illustrates our approach more clearly than simple bipartitioning.

As with the routing problems, we begin with a *redundant* solution. Unlike most move-based algorithms, in which a vertex is assigned to a single randomly chosen cluster, we assign each vertex to *multiple* clusters. If a vertex is assigned to more than one cluster, these assignments are *redundant*. After an initial assignment of vertices to multiple clusters, individual vertex assignments are removed in a greedy manner, until a final non-redundant solution is produced.

Figure 2 illustrates the iterative deletion approach. In this Figure, vertex v_2 is assigned to four distinct clusters, while connected vertices v_1 and v_3 each have a single cluster assignment. We select a single redundant assignment for v_2, and remove it; this selection is based on the locations and assignments of vertices connected to v_2. While we may wish to have the final assignment of v_2 to either cluster 1 or cluster 2, we do not commit to either at this step; rather, we remove from consideration either cluster 3 or cluster 4.

The multi-way formulation provides an important contrast to greedy clustering. While an operation in a clustering approach might merge v_2 into cluster 1 or cluster 2, the deletion approach simply specifies that v_2 will *not* be in either cluster 3 or cluster 4. After one operation, the clustering approach would have no freedom left to optimize the location of vertex v_2, while the iterative deletion approach would have three of four possible choices still available.

	Cluster 1	Cluster 2	Cluster 3	Cluster 4
v_1	1	0	0	0
v_2	1	1	1	1
v_3	0	1	0	0
e_1	2	1	1	1
e_2	1	2	1	1

Table 1. Links of vertices and hyperedges to clusters; the vertex v_2 is linked to multiple clusters, while vertices v_1 and v_3 are fixed. The hyperedges have links which are summations of the connected vertices, and both hyperedges are fixed.

We summarize the difference between traditional iterative improvement algorithms (such as the move-based FM algorithm) and the iterative deletion approach as follows. *Iterative improvement algorithms pursue moves that appear the "best," while iterative deletion algorithms eliminate moves that appear the "worst."* For bipartitioning, the iterative deletion approach is similar to a greedy clustering approach; the differences becomes apparent when we consider k-way partitioning.

4.2. Vertex Selection

Clearly, if we wish to determine which assignment is "worst," we must employ some sort of cost function. From an initial assignment, we can obtain an indication of which vertices and hyperedges are present in any given cluster. The hyperedge locations (or possible locations) allow the generation of costs on a per cluster basis, and these costs determine the cost of a vertex relative to a given cluster.

Figure 2 shows a simplified graph and partial assignment, while Table 1 shows our approach for vertex and hyperedge assignments to each cluster. Table 2 shows the resulting costs. Our current implementation utilizes an extremely simple cost calculation approach, which operates as follows.

- Each vertex contains a number of *links* to clusters. Each link is considered to be an assignment. When a vertex contains only a single link, we say that it has a *fixed* assignment to a cluster, or that the vertex is *fixed*. A fixed vertex cannot be moved, and will be placed in the cluster as part of the algorithm output.

  At initialization, $link(v_i, c_j) = 1$ for all vertices v_i, and all clusters c_j.

- Each connecting hyperedge also contains a number of *links*, which are simply the summation of the links of the vertices connected by the hyperedge. If a vertex of a hyperedge is fixed to a given cluster, we say that the hyperedge is also fixed to the cluster. If multiple vertices of the hyperedge are fixed to distinct clusters, the hyperedge is cut.

  For all edges, $link(e_k, c_j) = \sum_{v_i \in e_k} link(v_i, c_j)$

- The *cost* of a hyperedge is calculated on a per-cluster basis. Each cluster has different cost, based on the number of links to the cluster. If a hyperedge is fixed, it has high cost for a single cluster. If a hyperedge is cut, all clusters have a cost of 0.

  We calculate $cost(e_k, c_j)$ as a function of the links of component vertices. This function is described in more detail below.

- The *cost* of a vertex is also calculated on a per-cluster basis, and is simply the summation of costs of hyperedges to which the vertex belongs.

  $cost(v_i, c_j) = \sum_{v_i \in e_k} cost(e_k, c_j)$

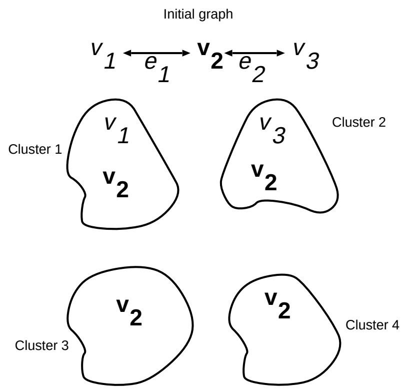

Figure 2. A redundant assignment of vertices to clusters. In this example, the vertex v_2 is assigned to four distinct clusters. Through the application of iterative deletion, the least desirable assignments for v_2 are removed, until a single assignment for v_2 remains. In this example, assignments of v_2 to clusters 1 and 2 are desirable, while assignments to clusters 3 and 4 are less desirable.

	Cluster 1	Cluster 2	Cluster 3	Cluster 4
e_1	k_2	0	0	0
e_2	0	k_2	0	0
v_2	k_2	k_2	0	0

Table 2. Cost calculation for hyperedges and vertices. Hyperedge e_1 has a preference for cluster 1, while hyperedge e_2 has a preference for cluster 2. The cost associated with vertex v_2 is derived from the costs of the hyperedges which connect to it. As vertices v_1 and v_3 are fixed, we do not need to calculate costs for them.

In our implementation we assign a cost of k_0 to hyperedges where vertices are completely unrestricted. When a vertex assignment is removed (and we have at least some preference regarding vertex assignments), we use a cost of k_1, and distribute it among the clusters to which links remain. When a vertex becomes fixed to a specific cluster, we use a cost of k_2, and have a cost of 0 for the remaining clusters. If the hyperedge is cut (vertices are fixed to more than one cluster), we use a cost of 0 for all clusters.

In our experiments, we use $k_0 = 0$, $k_1 = 1$, and $k_2 = 2$. We divide vertex costs by the degree of the vertex, and hyperedge costs by the cardinalty of the hyperedge, in an effort to prefer small local edges over larger ones (which are more likely to be cut). Experiments with other cost functions are in progress.

We have a preference for removing redundant assignments which are undesirable *relative to the other possible assignments for the vertex.* Rather than simply selecting the minimum cost element, we use the *difference between maximum and minimum assignment cost* to influence our selection. For example, if we have one vertex v_i, assigned to cluster c_m with cost 10, and to cluster c_n with cost 11, and a second vertex v_j has costs 10 and 20, we will prefer to make a selection regarding v_j (and we prefer to remove v_j from cluster c_m).

In order to ensure balanced cluster sizes, we iteratively select the best candidate within the cluster of highest weight. In bipartitioning applications, this results in an alternation between clusters. We currently consider only exact or nearly exact partitioning (cluster sizes differ by at most one vertex).

4.3. Random Seeds

In most implementations of the iterative improvement move-based FM algorithm, optimization begins with a randomly generated partition. By repeating the optimization, differing results can be obtained.

We apply a similar technique here; for k-way partitioning, we select k vertices, and assign each to one cluster. From this initial assignment, neighboring hyperedges and vertices are biased towards certain clusters. This random seeding produces a range of solutions, and we select the best result observed.

4.4. Efficient Implementation

As is done by the FM algorithm, we repeatedly select one element from the set of all vertices at each step. The FM algorithm owes its efficiency to a simple gain bucket strategy, and we utilize a similar scheme. Using this strategy, element selection can be performed in constant time.

Cost updates are also constant time in practice. For partitioning problems where the hypergraph is based on a circuit netlist, we can expect a small upper bound on the cardinality of any vertex, or the degree of any hyperedge.

The number of vertex assignments that must be removed is linear with the number of vertices, multiplied by the number of clusters that we consider. If we are performing k-way partitioning on a hypergraph with n vertices, we must remove $(k - 1) * n$ assignments.

4.5. Summary

A fundamental difference between the approach presented here and traditional partitioning approaches is that we begin with *redundant* assignments of vertices to clusters, and remove them. Conceptually, this is similar to the process of team construction in professional sports. During a *training camp* many players are considered for each position on the team; poor players are eliminated from consideration one at a time, until the remaining players constitute the final team.

5. EXPERIMENTAL RESULTS

A primary objective of this work is to allow a direct and clear comparison of the iterative deletion approach to the more common iterative improvement approach. Thus, we are interested in flat bipartitions and multi-way partitions. State of the art partitioners employ a number of techniques which improve results, obscuring the relative merits of each algorithmic approach.

As we are interested in partitioning as it relates to placement problems, we apply our approach to the recently introduced ISPD98 benchmarks[1]. We consider both the metric of minimum cut, and minimum weighted cut (where we have a variety of differing net weights).

5.1. Minimum Cut Experiments

In much of the previous work on hypergraph partitioning, the cost objective is minimum cut, with each hyperedge contributing cost 0 if uncut, or 1 if cut. We focus on this cost metric in this subsection.

5.1.1. Bipartitioning Results

To compare the performance of the iterative deletion approach to that of bipartitioning algorithms, we utilize an implementation of the FM[10] algorithm made available by Charles Alpert. This implementation was developed by Shantanu Dutt and Wenyong Deng, and was later improved by Charles Alpert and Andrew B. Kahng. The FM algorithm provides a simple iterative improvement approach, allowing a direct comparison.

In Table 3, we compare a single pass of our iterative deletion partitioning (IDP) algorithm to multiple passes of the FM algorithm. We also apply the FM algorithm as a postprocessing step (IDP + FM), using the iterative deletion solution as a starting point (rather than a random vertex assignment). For our bipartitioning experiments, we ran our algorithm 20 times, each time with a different seed. The IDP results reported are the minimum, maximum, and average cut sizes for *exact* bipartitions; other results allow a 5% variation in cluster size. Run time is roughly linear with problem size: the largest example, ibm18, requires just under two minutes, running on a 233Mhz Pentium-based laptop computer. The FM algorithm was also run 20 times, using different random seeds. We also include the results of a state of the art partitioning algorithm, hMetis[13] version 1.5.

While our iterative deletion based algorithm produces higher cut sizes, the solution obtained can serve as a good starting point for FM-based improvement. In these benchmarks, the solution quality obtained by using FM as a postprocessing phase were comparable to the quality obtained by FM alone, but the number of improvement passes required was roughly halved.

The quality of solution obtained by iterative deletion is comparable to that of an iterative improvement which is “half way” through a series of optimization passes.

5.1.2. Multi-Way Partitioning Results

To compare performance on flat multi-way partitioning problems, we use the results of the K-FM algorithm[15] and K-PM algorithm[5], as reported in [5]. The K-FM algorithm provides a simple iterative improvement approach, allowing a direct comparison with the iterative deletion approach. Results of a state of the art partitioning algorithm, hMetis[13], are included as well; this algorithm employs multilevel clustering. These results are presented in Table 4.

Again, we apply only a *single* pass of iterative deletion. Surprisingly, the linear time greedy implementation of iterative deletion outperforms the K-FM algorithm, which employs multiple passes of optimization. We run our IDP algorithm 20 times for each benchmark, while the K-FM results are from 50 runs. The abundance of possible choices apparently causes the iterative improvement approach to become trapped in a local minima early in the optimization process. The K-PM algorithm is run 20 times.

In many respects, the K-FM algorithm of [15] is something of a “straw man” multi-way partitioning approach; current approaches to multi-way partitioning would involve multilevel clustering or a reduction in the types of optimization moves considered. The results here are interesting, however, as they clearly show that a generic iterative improvement based approach has considerably more difficulty than an iterative deletion based approach on a well studied problem. For optimization problems in which we have many possible choices, and iterative improvement methods provide poor performance, iterative deletion based approaches may be quite effective.

We are currently adapting our algorithm to produce output in a format that can be utilized by the K-PM algorithm, and expect that post processing of the iterative deletion solution will be comparable or superior to that of K-PM alone, but will require a substantial reduction in the number of optimization passes.

5.2. Weighted Minimum Cut Experiments

For performance driven placement applications, it is desirable to model signal delay as part of our optimization objective. In deep submicron design, delay is not linear with wire length, so we can expect a large range of values. If we apply partitioning algorithms to placement problems, a formulation which addresses weighted edge cost may be more appropriate.

In this subsection, our objective is the minimization of the total weight of cut hyperedges. The FM implementation of Dutt and Deng does not support weighted hyperedges; for comparison, we use our own implementation of the FM algorithm. Performance (in terms of total cut sizes) of our FM implementation is comparable to that of Dutt and Deng for the non-weighted objective.

To evaluate the impact of differing hyperedge weights on algorithm performance, we applied FM, IDP, IDP with FM postprocessing, and the hMetis algorithm (version 1.5.3) to the benchmarks, using net weights that ranged from 95 to 105. The weight of a hyperedge e_i is $95 + i \; mod$ 11; there are 11 distinct weights, with the average edge weight being roughly 100 (we divide weights reported by 100 to make them comparable to the unit-cost results). Table 5 shows the results of these experiments; as our FM implementation is substantially slower than that of Dutt and Deng, we limit our experiments to ten of the smaller benchmarks.

As would be expected by the observations in [11], the performance of the FM algorithm degrades substantially. The recommended Last-In-First-Out (LIFO) strategy has little effect, as the wide range of net weights results in extremely few ties for the the FM algorithm. For variable edge weight problems, a single pass of iterative deletion outperforms iterative improvement; using the iterative deletion result as a starting point, iterative improvement is able to obtain a solution that is much better than with iterative improvement or iterative deletion alone. The multilevel clustering approach of hMetis continues to produce extremely good solutions, obtaining results that are nearly identical to those of the unit-cost experiments.

6. CONCLUSION

In this paper, we have presented a new partitioning algorithm based on iterative deletion. As it supports multi-way partitioning, does not require extensive clustering, runs in linear time, and can support complex cost functions, it is relevant to circuit placement problems.

We have provided a comparison between the iterative deletion approach, and the widely used iterative improvement approach. We observe that each has areas in which they perform well. Iterative improvement performs well in bipartitioning, while iterative

	IDP			FM				IDP + FM				hMetis
		Cut			Cut				Cut			
	Min	Max	Avg	Min	Max	Avg	Passes	Min	Max	Avg	Passes	Cut
ibm01	854	1290	1037.6	264	662	483	11.9	270	627	501.6	6.0	180
ibm02	802	2289	1551.1	276	820	443.1	11.2	276	652	452.7	6.1	262
ibm03	2316	3240	2730.7	1359	3491	2200.5	18.6	1665	2270	1985.0	6.4	956
ibm04	2964	4683	3724.7	739	2789	1228.5	18.1	682	2533	1120.9	11.2	542
ibm05	3991	5973	4984.3	2034	4007	2880.3	29.1	1989	3755	3011.7	9.4	1715
ibm06	2084	3770	2873.9	1027	2321	1364.2	18.4	1017	2004	1475.2	9.0	888
ibm07	2429	4944	3872.9	1044	3687	2342.7	20.4	1260	3365	2209.3	6.7	853
ibm08	3354	5448	3861.6	1317	4644	2725.9	23.2	1304	3764	2035.3	8.9	1142
ibm09	2294	5236	3422.8	1331	3696	2591.3	20.8	983	3492	1813.4	8.1	624
ibm10	6709	8261	7070.9	2211	3680	2837.3	18.4	2239	3931	3279.9	10.7	1256
ibm11	5226	7587	6483.2	2391	7678	4139.8	22.1	1543	4157	2922.2	14.1	960
ibm12	8681	12492	10869.0	2491	5788	3463.5	21.4	3257	5996	4428.2	14.1	1918
ibm13	4489	9112	6618.6	1272	3733	2243.2	17.4	1662	4479	2660.3	9.7	840
ibm14	8666	13206	11704.6	2876	11806	7144.5	26.1	3015	7431	5329.3	11.4	1837
ibm15	8794	12993	10603.1	4576	11509	8435.1	19.6	4719	7937	6266.7	10.9	2625
ibm16	12723	17943	14885.4	2279	10109	5933.1	20.2	2832	10728	5213.5	10.1	1755
ibm17	22021	28963	25423.3	5316	14258	8509.4	29.4	3834	11811	6971.3	18.9	2238
ibm18	5795	15730	10957.6	1791	5003	3254.8	26.0	1707	5638	4115.4	13.3	1541
Ratio	4.41	7.33	5.79	1.52	4.41	2.72		1.50	3.80	2.47		1.00

Table 3. Bipartitioning results for industry benchmarks. We compare runs of a single pass of our iterative deletion partitioning (IDP) approach to those of a complete (multiple pass) run of an FM-based algorithm. We also consider the effects of applying FM as a postprocessing step to the initial iterative deletion solution (IDP+FM). The results of the recent multilevel clustering based partitioner, hMetis, are also included. The final row shows the average performance relative to that of hMetis.

	4-Way Partitioning						8-Way Partitioning					
	IDP			K-FM	K-PM	hMetis	IDP			K-FM	K-PM	hMetis
		Cut						Cut				
	Min	Max	Avg	Cut	Cut	Cut	Min	Max	Avg	Cut	Cut	Cut
ibm01	1709	2740	2251.2	3212	576	496	3256	3842	3577.6	4234	857	755
ibm02	2625	5175	4150.1	5984	688	615	6611	7596	7118.9	7138	2069	1874
ibm03	4112	5288	4644.8	6737	2596	1682	6217	7651	6929.6	8263	3512	2396
ibm04	6117	7467	6655.6	8332	2290	1711	9116	10687	9606.0	10347	3751	2782
ibm05	6242	8694	7755.9	8537	4225	3040	8956	10093	9474.7	9387	5760	4443
ibm06	4500	6031	5244.0	8664	2096	1592	6537	8403	7751.0	10923	2954	2257
ibm07	5923	8302	7437.2	12724	3069	2168	10300	12091	11222.4	15725	4375	3284
ibm08	7451	9766	8718.1	12845	2945	2426	11528	13625	12476.6	16056	4532	3462
ibm09	7109	9821	8522.8	15888	2838	1685	13401	15549	14493.4	19619	4759	2664
ibm10	11178	16454	13611.5	20820	3163	2280	17880	21269	19730.0	26170	4888	3799
ibm11	9753	14488	12090.4	21448	4685	2300	16900	19138	17996.8	27479	6059	3543
ibm12	14762	18405	16777.7	23081	5258	3799	21278	25248	23338.2	28764	7946	6024
ibm13	11712	18785	15618.4	24758	3102	1760	19294	25802	22883.2	30975	4390	2858
ibm14	18989	26303	22483.8	38767	6451	3249	27873	36808	31513.2	49334	8424	4795
ibm15	19621	29655	24080.8	48130	8310	5014	30530	39930	36607.6	64235	11465	6610
ibm16	21134	36183	28970.0	54578	6228	3847	41441	51057	45358.1	65553	10372	6203
ibm17	35757	44883	40541.9	64340	9326	5398	50365	56957	53500.6	75432	14733	8695
ibm18	23066	30743	26997.5	53128	3952	2872	38854	45069	42234.6	65361	6588	5210
Ratio	4.35	6.29	5.36	8.81	1.51	1.00	4.46	5.39	4.93	6.80	1.45	1.00

Table 4. Multi-way partitioning results for large benchmarks. We compare the iterative deletion partitioning (IDP) solution (without postprocessing) to the flat multi-way partitioning algorithm K-FM, the pair-wise matching partitioner K-PM, and the state of the art multilevel clustering approach of hMetis. The final row shows the average performance relative to that of hMetis.

	IDP			FM			IDP+FM			hMetis
	min	max	avg	min	max	avg	min	max	avg	
ibm01	1393.32	1865.67	1664.06	1498.46	2266.44	1834.97	676.73	1047.83	885.47	181.48
ibm02	2250.44	3742.90	2834.55	725.83	2193.68	1341.71	786.24	1709.22	1116.31	261.28
ibm03	4355.60	5530.26	4865.17	3916.18	4746.11	4378.61	2244.79	3110.86	2558.95	955.62
ibm04	5560.26	6248.49	5804.49	2640.45	5187.13	4290.68	2935.87	3735.49	3258.18	541.81
ibm05	5996.70	7535.68	6766.83	4770.47	6739.43	6127.14	3266.82	4903.49	4153.86	1760.51
ibm06	5754.91	6860.46	6206.89	2132.42	6194.21	5392.24	2821.29	3731.58	3170.51	922.75
ibm07	6267.37	7835.69	7269.60	7039.75	8486.00	7591.87	2908.03	3884.71	3530.42	883.54
ibm08	6656.86	8817.66	7801.13	8252.02	9206.45	8874.09	3100.73	4708.88	4101.97	1143.49
ibm09	8456.28	9359.26	8851.69	9460.10	10748.72	10061.20	4190.30	4833.80	4439.38	656.30
ibm10	9534.35	12222.33	10197.81	9663.52	12342.00	10707.06	4657.03	7346.20	5414.12	1264.73
Ratio	7.41	9.41	8.30	6.23	8.98	7.72	3.55	5.10	4.23	1.00

Table 5. Variable weight bipartitioning results. We compare the iterative deletion partitioning solution to that of a traditional FM algorithm, and to the combination of Iterative Deletion with post-processing by FM. We include also the result of the state of the art partitioning algorithm, hMetis. Hyperedge weights vary from 0.95 to 1.05 (with an average weight of 1 across all nets). The final row shows the average performance relative to that of hMetis.

deletion performs well in multi-way partitioning, and with variable edge weights. The performance of iterative improvement approaches can be enhanced through the use of multilevel clustering and a restriction of optimization moves considered; we anticipate that similar enhancements can be made to our iterative deletion based approach.

With a single linear time optimization pass, iterative deletion was able to outperform multiple passes of iterative improvement. This suggests that the iterative deletion approach is able to make many decisions that are of good quality from a "global" perspective, particularly when many choices are available. For the multi-way partitioning problem, in which we have an extremely large number of choices, the iterative deletion based approach seems less susceptible to poor quality local minima. For modern circuit design problems, where we are faced with large numbers of constraints and many optimization choices, iterative deletion may prove to be extremely effective.

We are currently adapting our algorithm to produce output that can be utilized by the K-PM algorithm of [5]. In our experiments with bipartitioning, post-processing by the FM-based algorithm[10] improved performance substantially, while the "good starting point" produced by our iterative deletion approach reduced the number of optimization passes required. We expect similar results through post-processing with K-PM.

Alternative cost functions and methods to apply multiple passes of iterative deletion are currently under consideration. If we examine a single pass of an iterative improvement algorithm such as FM, solution quality is generally extremely poor; multiple passes may improve the performance of iterative deletion substantially. While some progress has been made on these issues, work is incomplete at present.

We plan to adapt this approach for use in large circuit placement applications. As our formulation can consider routing distances or signal delay easily, we can produce rough circuit placements quickly, and then refine them with more traditional local optimization algorithms.

Acknowledgements: the author would like to thank Prof. Jason Cong, the members of the UCLA VLSI CAD Lab, and the reviewers for their constructive comments.

REFERENCES

[1] C. J. Alpert. The ispd98 circuit benchmark suite. In *Proc. Int. Symp. on Physical Design*, pages 80–85, 1998.

[2] C. J. Alpert, J.-H. Huang, and A. B. Kahng. Multilevel circuit partitioning. In *Proc. Design Automation Conf*, pages 530–533, 1997.

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

[4] M. A. Breuer. A class of min-cut placement algorithms. In *Proc. Design Automation Conf*, pages 284–290, 1977.

[5] J. Cong and S. K. Lim. Multiway partitioning with pairwise movement. In *Proc. Int. Conf. on Computer Aided Design*, pages 512–516, 1998.

[6] J. Cong and P. H. Madden. Performance driven global routing for standard cell design. In *Proc. Int. Symp. on Physical Design*, pages 73–80, 1997.

[7] J. Cong and B. Preas. A new algorithm for standard cell global routing. *Integration, the VLSI Journal*, 14:45–65, 1992.

[8] J. Cong and M. Smith. A parallel bottom-up clustering algorithm with applications to circuit partitioning in vlsi designs. In *Proc. Design Automation Conf*, pages 755–760, 1993.

[9] A. E. Dunlop and B. W. Kernighan. A procedure for placement of standard-cell vlsi circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, CAD-4(1):92–98, 1985.

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

[11] L. W. Hagen, D. J.-H. Huang, and A. B. Kahng. On implementation choices for iterative improvement partitioning algorithms. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16(10):1199–1205, 1997.

[12] D. J.-H. Huang and A. B. Kahng. Partitioning-based standard cell global placement with an exact objective. In *Proc. Int. Symp. on Physical Design*, pages 18–25, 1997.

[13] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. In *Proc. Design Automation Conf*, pages 526–529, 1997.

[14] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning of electrical circuits. *Bell Systems Technical Journal*, pages 291–307, 1970.

[15] L. A. Sanchis. Multiple-way network partitioning with different cost functions. *IEEE Trans. on Computers*, 42(22):1500–1504, 1993.

[16] D. Sylvester and K. Keutzer. Getting to the bottom of deep submicron. In *Proc. Int. Conf. on Computer Aided Design*, pages 203–211, 1998.