On Implementation Choices for Iterative Improvement Partitioning Algorithms*

Lars W. Hagen†, Dennis J.-H. Huang and Andrew B. Kahng

UCLA Department of Computer Science, Los Angeles, CA 90024-1596
† Cadence Design Systems, Inc., San Jose, CA 95134

Abstract

Iterative improvement partitioning algorithms such as those due to Fiduccia and Mattheyses (FM) [2] and Krishnamurthy [5] exploit an efficient gain bucket data structure in selecting modules that are moved from one partition to the other. In this paper, we investigate three gain bucket implementations and their effect on the performance of the FM partitioning algorithm. Surprisingly, selection from gain buckets maintained as Last-In-First-Out (LIFO) stacks leads to significantly better results than selection from gain buckets maintained randomly (as in [5] [7]) or as First-In-First-Out (FIFO) queues. Our experiments show that LIFO buckets result in a 35% improvement over random buckets and a 42% improvement over FIFO buckets. Furthermore, eliminating randomization from the bucket selection is of greater benefit to FM performance than adding the Krishnamurthy gain vector. By combining insights from the LIFO gain buckets with those of Krishnamurthy's original work, a new higher-level gain formulation is proposed. This alternative formulation results in a further 16% reduction in the average cut cost when compared directly to the Krishnamurthy formulation for higher-level gains, assuming LIFO organization for the gain buckets.

1 Preliminaries

This paper discusses the problem of bipartitioning a circuit netlist hypergraph $G = (V, E)$, where the set of modules V is divided into disjoint U and W to minimize the number of signal nets from E that cross the cut. In production software for circuit partitioning, *iterative improvement* is a nearly universal approach, either as a postprocessing refinement to other methods or as a method in itself. Iterative improvement is based on *local* perturbation of the current solution and can be either greedy (the Kernighan-Lin method [3] [9] and its algorithmic speedups by Fiduccia and Mattheyses [2], Krishnamurthy [5] and Dutt [1]) or hill-climbing (the simulated annealing approach of Kirkpatrick et al. [4], Sechen [10] and others). Virtually all implementations will also use multiple random starting configurations ("multi-start") [6] [11] in order to yield predictable performance ("stability").

This paper will focus on iterative improvement algorithms which are based on the greedy strategy: start with a current feasible solution and iteratively perturb it into another feasible solution, adopting the perturbation as the next solution only if it improves the cost function. The type of perturbation used determines a topology over the set of feasible solutions, known as a *neighborhood structure*. For the cost function to be "smooth" over the neighborhood structure, the perturbation (also known as a *neighborhood operator*) should be small and "local". Usual neighborhood operators for graph/circuit partitioning involve swapping a pair of modules or shifting a single module across the cut. Early greedy improvement methods apply such operators, and quickly find local minima which usually correspond to poor solutions.

In 1970, Kernighan and Lin [3] introduced what is often described as the first "good" graph partitioning heuristic. The Kernighan-Lin (KL) algorithm uses pair-swapping, and proceeds in *passes*. During each pass, every module is moved exactly once. At the beginning of the pass, all modules are "unlocked" and the *gain* (i.e., the decrease in cut nets that would result from moving a given module to the other partition) is calculated for each of the $n = |V|$ modules. Then, the pair of unlocked modules in U and W with highest combined gain is found by searching through the $O(n^2)$ possible pairs. After the selected modules are swapped, they become "locked" and the algorithm updates both the cost of the new partition and the gains of the remaining unlocked modules. This process is iterated until all the modules are locked, at which point the lowest-cost partition encountered over the entire pass is restored and returned. Another pass is then executed using the result from the previous pass as its starting point; the algorithm terminates when a pass fails to improve the cost function. The advantage of the KL algorithm over greedy pair-swapping is that it is in some sense able to move out of local minima. This occurs because the pair of modules with highest combined gain is always swapped, even if this combined gain is negative. However, if we consider all the

*This research was supported in part by NSF grant MIP-9257982 and matching funds from High-Level Design Systems.

solutions that are reachable within a single pass of the algorithm to be "neighbors" of the starting solution, then the KL algorithm is still greedy.

The main disadvantages of the KL algorithm, as presented in [3], were (i) that it only works on graphs and (ii) that it is computationally expensive. Although the number of passes in most cases is relatively low, the KL algorithm requires evaluation of $O(n^2)$ swaps before every move, resulting in a complexity per pass of $O(n^2 \log n)$. Schweikert and Kernighan [9] extended KL to hypergraphs, but did not improve the time complexity of the algorithm.[1]

The FM Algorithm

In 1982, Fiduccia and Mattheyses [2] presented a KL-inspired algorithm which reduced the time per pass to linear in the size of the netlist (i.e., $O(p)$ where p is the total number of pins). The Fiduccia-Mattheyses (FM) algorithm is very similar to KL: (i) FM also performs passes within which each module is moved exactly once; (ii) FM also records all solutions encountered during the pass and returns the best one; and (iii) FM also continues to perform passes until a pass fails to improve the cost function.

The primary difference between the KL and FM algorithms lies in the neighborhood operator. Instead of swapping a pair of modules, FM moves a single module at a time. In other words, the gain lists are searched for a single module which has highest gain. This subtle change allows for a significant improvement in runtime with little loss in solution quality. Fiduccia and Mattheyses amortize the cost of updating the module gains, such that the total cost of finding the highest-gain module is $O(p)$ per pass. The enabling data structure is an array of "gain buckets" which groups the modules of a given partition according to their gains.

Krishnamurthy's Extension to FM

Over the past decade, FM has become perhaps the single most widely used and cited partitioning algorithm in the VLSI CAD area. Many works have investigated possible improvements and extensions. One commonly-cited extension is that of Krishnamurthy [5], who showed how one could efficiently introduce "look-ahead" into the FM algorithm to improve tie-breaking when the highest-gain bucket contains more than one module. Specifically, Krishnamurthy extends the gain value of a module into a gain *vector* which gives a sequence of potential gain values corresponding to various numbers of moves into the future.

Krishnamurthy defines the *binding number* $\beta_U(s)$ of signal net s with respect to partition U to be the number of unlocked modules of s in partition U, unless there is a locked module of s in partition U, in which case $\beta_U(s) = \infty$. Intuitively, the binding number $\beta_U(s)$ is a measure of how difficult it is to move net s out of partition U. The binding number $\beta_W(s)$ is similarly defined. The k-th level gain $\gamma_k(v_i)$ of module $v_i \in U$ is then given by[2]

$$\gamma_k(v_i) = |\{s \in E | v_i \in s, \beta_U(s) = k, \beta_W(s) > 0\}| - |\{s \in E | v_i \in s, \beta_U(s) > 0, \beta_W(s) = k-1\}|$$

Each element $\gamma_k(v_i)$ in the gain vector corresponds to the k-th level gain of module v_i. Note that the first-level gain $\gamma_1(v_i)$ corresponds to the gain used in the FM algorithm.

The intuition behind the higher-level gains γ_k, with $k > 1$, is that the positive term counts the number of nets which will have binding number $k-1$ after the move, while the negative term counts the number of nets with current binding number $k-1$ which will have binding number equal to ∞ after the move. In other words, the positive term counts nets with binding number $k-1$ that are "created" by the move; the negative term counts nets with binding number $k-1$ that are "destroyed" by the move. (The created nets lie on the side that the module is moving "from", and the destroyed nets lie on the side that the module is moving "to".) Krishnamurthy's method uses lexicographic ordering of the vectors (γ_1, γ_2, γ_3, ...) to break ties when an FM gain bucket contains more than one module. Krishnamurthy compared his FM plus higher-level gain (FM+HL) algorithm with the original FM algorithm and found that adding second- and third-level gains improved the average solution quality, with an added computational expense of only $O(kp)$, where k is the number of values maintained in (i.e., the size of) the gain vector. This was confirmed by Sanchis [7], who extended FM+HL to multi-way partitioning.

2 Tie-Breaking in the FM Algorithm

During a typical pass in the FM algorithm, there are usually many ties (i.e., the highest-gain bucket will contain more than one module). Figure 1 shows the number of modules in the highest-gain bucket at each move throughout the first pass of FM for the Primary1 test case (we plot the average and maximum over 1000 runs). This section investigates how the method used

[1] The reduction from $O(n^3)$ to $O(n^2 \log n)$ is achieved by maintaining a sorted list of costs. Recently, Dutt [1] presented a speedup of the original KL algorithm, called QuickCut, which uses an improved data structure such that only $O(d^2)$ node pairs need to be examined to find the pair with maximum gain (d is the maximum node degree). As a result, QuickCut has time complexity of only $O(\max(ed, e\log(n)))$, where e is the number of edges in the graph. QuickCut currently works only on graphs, but an extension to hypergraphs seems possible.

[2] The notation used for the Krishnamurthy formulae are adapted from [5]. Note that in order to handle 1-pin nets correctly, the term $\beta_U(s) > 0$ should to be changed to $\beta_U(s) > 1$. However, 1-pin nets can also be eliminated while reading in the netlist, obviating the need for such a change.

to choose a cell (i.e., module) from the highest-gain bucket, and the method used to add updated cells into gain buckets, will together affect the performance of the FM algorithm.

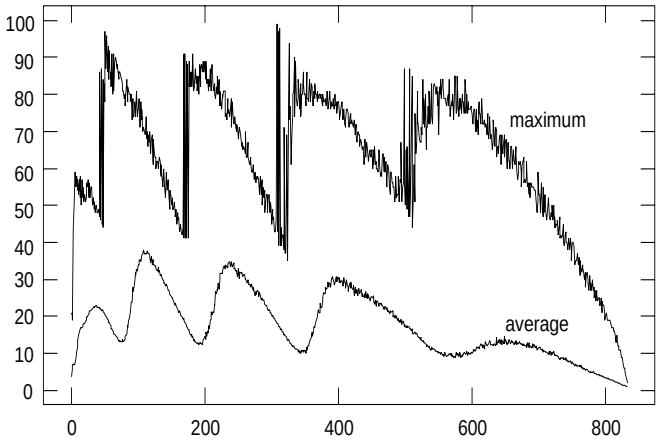

Figure 1: Number of modules in the highest-gain bucket during the first pass of FM for test case Primary1. The average and maximum numbers were generated from 1000 separate FM runs.

2.1 Tie-Breaking Schemes

In the original paper describing the FM algorithm [2], the gain buckets consist of doubly-linked lists. To identify the cell to move, Fiduccia and Mattheyses consider the first cell in the highest-gain bucket of each partition. Figure 2 reproduces the "MAXGAIN" bucket used in the algorithm description of [2]; note that this bucket has only a pointer to the head of the list. Thus, it is reasonable to infer that when selecting a cell to move from the highest-gain bucket, Fiduccia and Mattheyses selected the cell at the head of the list. Such an inference is supported by the fact that this operation must be performed in $O(1)$ time in order for the complexity of the algorithm to remain at $O(p)$. Choosing the first cell in the list satisfies this complexity requirement. With respect to inserting an updated cell into a new bucket, [2] removes a cell from its current list and moves it to the head of its new bucket list. Considering the removal and insertion procedures together, we see that the gain buckets function as LIFO stacks (remove at head, insert at head), but could just as easily function as FIFO queues (remove at head, insert at tail) if a pointer to the tail of the list is incorporated into the data structure. Fiduccia and Mattheyses do not discuss the implications of the choice of bucket organization on their algorithm's performance. However, as we shall show, this choice has a significant effect.

Interestingly, neither Krishnamurthy nor Sanchis points out any change in the tie-breaking heuristic used to select among cells with identical higher-level gains. The natural inference is that their works also

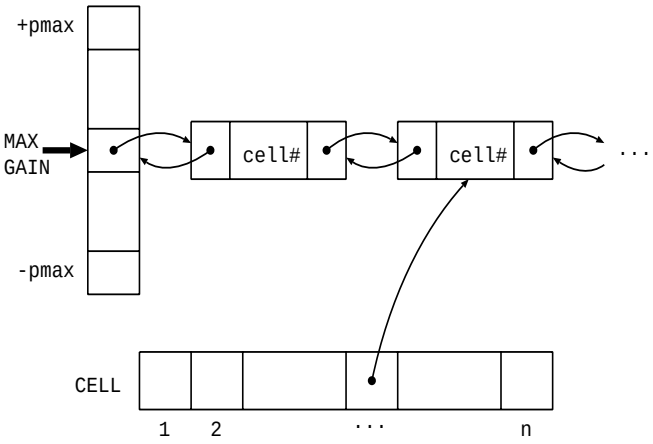

Figure 2: The gain bucket list structure as shown in [2].

use a LIFO mechanism, following the original FM algorithm description. However, in the code distributed by Sanchis [8], it is clear that the highest-gain module is selected *randomly* in the event of ties. Sanchis in [7] never discusses the consequences of this change, but writes: "We also randomized arbitrary choices in the algorithm and performed a number of runs on each network partition at each different level" (page 68, right column, first paragraph). A similar statement is made by Krishnamurthy [5] (page 442, left column, first paragraph): "We observe that a single run would not provide sufficient evidence to compare the results; for each of these algorithms, being heuristic in nature involves making certain arbitrary choices, usually in the form of selecting any one element from a set containing more than one element. Thus, we randomized such arbitrary choices and performed a number of runs." That possibly both of these works introduce random tie-breaking in comparing FM+HL to the original FM is not trivial. Randomization not only increases the time complexity of the algorithm but, more critically, places into question the conclusions drawn from the experimental results.

We have examined how the "obvious" tie-breaking mechanism proposed by Fiduccia and Mattheyses [2] compares with alternative schemes. In particular, we have compared LIFO selection with random selection (used by Sanchis [7]) and FIFO selection (an alternative organization with complexity similar to LIFO). Our testbed is the code distributed by Sanchis [8] with appropriate modifications made for handling LIFO and FIFO selection.

In all of our experiments, we assume each node has unit area, and we constrain the partition sizes $|U|$ and $|W|$ to differ by at most 1.

2.2 Experimental Results

The third column of Table 1 clearly shows the effects of the selection methodology.[3] Surprisingly, the FIFO scheme is no better than random selection. The LIFO scheme gives considerable improvement over random selection. One possible explanation may be that organizing the buckets such that "most recently visited" modules are placed near the beginnings of the gain buckets implicitly causes neighborhoods (or perhaps clusters) of modules to be moved together. Furthermore, since there are two gain structures, one for each partition, it is possible for each partition to "pull" on different clusters while maintaining the balance. If these clusters are non-interfering, i.e., widely separated, more of the early moves will result in positive gain, enabling the current pass to reach a lower-cost point in the solution space. In other words, within each pass the solution cost curve will have a relatively sharper decline, and stay at lower costs as it returns back to the initial cost.[4]

Ckt (#nodes)	Method	# Krishnamurthy levels $k=1$	$k=2$	$k=3$	$k=4$
Prim1 (833)	LIFO	83	80	79	80
	RAND	108	94	86	84
	FIFO	125	97	91	88
struct (1952)	LIFO	68	66	65	66
	RAND	185	160	163	154
	FIFO	202	194	193	181
Prim2 (3014)	LIFO	296	281	279	275
	RAND	441	352	314	301
	FIFO	507	385	341	315
biomed (6514)	LIFO	172	181	199	220
	RAND	450	382	374	371
	FIFO	503	401	386	381
ind2 (12637)	LIFO	757	744	775	760
	RAND	1671	1299	1108	977
	FIFO	1766	1342	1188	1021
ind3 (15433)	LIFO	755	799	797	815
	RAND	2382	1164	1012	952
	FIFO	3285	1452	1151	1154
avq.sml (21918)	LIFO	835	911	950	964
	RAND	1507	1403	1384	1388
	FIFO	1842	1579	1499	1489
avq.lrg (25178)	LIFO	1140	971	1012	1030
	RAND	1837	1582	1527	1510
	FIFO	2193	1867	1768	1742
% Impr. vs. RAND	LIFO	48.3	36.8	30.3	26.4
	RAND	0	0	0	0
	FIFO	-17.1	-12.2	-10.1	-9.7

Table 1: Average cutsize results for 100 runs of FM (column 3) and Krishnamurthy higher level gains (columns 4-6) using LIFO (Last-In-First-Out), random and FIFO (First-In-First-Out) organization schemes for the gain buckets.

[3] For space reasons, all tables give average cutsize results over 100 runs. Minimum cutsize results are qualitatively similar and are separately available.

[4] Note that for bipartitioning, the cost at the end of the pass is exactly the same as the cost at the beginning of the pass, meaning that improvement results from an initial decrease in cost during the pass, followed by a corresponding increase in cost later in the pass.

Columns 4-6 of Table 1 show the effects of LIFO, random and FIFO selection on higher-level gains as defined by Krishnamurthy [5]. Introducing second-level ($k = 2$) gain and in some cases third-level ($k = 3$) gain seems to improve the solution quality for random and FIFO selection. For LIFO selection, we note the following:

- For constant k, the LIFO results are consistently better than the random or FIFO results.
- For each of the test cases, the $k = 1$ (FM) results using LIFO selection are significantly better than the results for any k using random or FIFO selection. In other words, the gain bucket organization has a greater effect on solution quality than the number of Krishnamurthy gain elements considered.
- For some large test cases (biomed, industry2 and avq.small), the $k = 1$ (FM) results are better than the $k > 1$ results under the LIFO scheme. Recall that the Krishnamurthy gain formula favors a module in a net that is locked to the side the module is moving to, and disfavors a module in an unlocked net having few modules on the side the module is moving to. In some sense, the LIFO organization has a similar function but with no penalty for moving a module that belongs to unlocked nets. That Krishnamurthy gains occasionally perform worse than LIFO FM suggests that following previously moved modules (i.e., moving to the side to which a net is locked) is more important than "staying away from the minority" (i.e., not moving to the side having very few modules of the incident nets).

3 A Krishnamurthy Variant

The above observation – that it may be more important to move modules which are incident to locked nets – suggests an alternative multi-level gain formulation. If a net is cut, and only one partition contains locked modules incident to this net, we will give higher priority to the modules in the partition with no locked modules incident to the net. We can implement this by increasing the gain elements of a module each time it is incident to a net which becomes locked to the opposite partition. For instance, assume module a is being evaluated for a move from partition U to partition W. If a net which contains module a has at least one module locked in partition W, and only free modules in partition U, we will increase all k-th level gains by 1, where $k \geq 2$. We avoid changing the first-level gain since this should always reflect the "actual" gain resulting from a move of this module. However, we choose to add 1 to all the other gain levels in order to make sure that the increased priority will have an effect on all tie-breaking instances.

Our alternative gain formulation can be expressed as follows for $k \geq 2$:

$$\gamma_k(v_i) = \\ |\{s \in E | v_i \in s, \beta_U(s) = k, \beta_W(s) > 0\}| - \\ |\{s \in E | v_i \in s, \beta_U(s) > 0, \beta_W(s) = k-1\}| + \\ |\{s \in E | v_i \in s, 0 < \beta_U(s) < \infty, \beta_W(s) = \infty\}|$$

The first two terms are identical to the formulation used by Krishnamurthy [5]. The third term is new and represents the "attraction" to locked modules. Figure 3 compares the Krishnamurthy gain vector with the gain vector resulting from our new formulation. In the beginning, an uncut net contains modules a, b, c, d and e and both gain vectors for module e are $(-1, 0, 0, 0, 1)$. After module a is moved to the other partition and becomes locked, the gain vector of module e is changed to $(0, 0, 0, 1, 0)$ in Krishnamurthy's formulation, but is changed to $(0, 1, 1, 2, 1)$ in our formulation. When we reach the case where module e is the only remaining module (case 5), the gain vectors are $(1, 0, 0, 0, 0)$ and $(1, 1, 1, 1, 1)$ for Krishnamurthy's and our formulations, respectively. Note that in this last case, the Krishnamurthy gain vector will not distinguish between module e and some other module x having gain vector $(1, 0, 0, 0, 0)$, where none of the nets incident to x have locked modules. By contrast, our gain formulation distinguishes between modules e and x since module x will have gain vector $(1, 0, 0, 0, 0)$ in our formulation. This is arguably an important difference: in most cases one would prefer to "uncut" the locked net incident to module e before committing the unlocked net incident to module x. Our experimental results also seem to support this view.

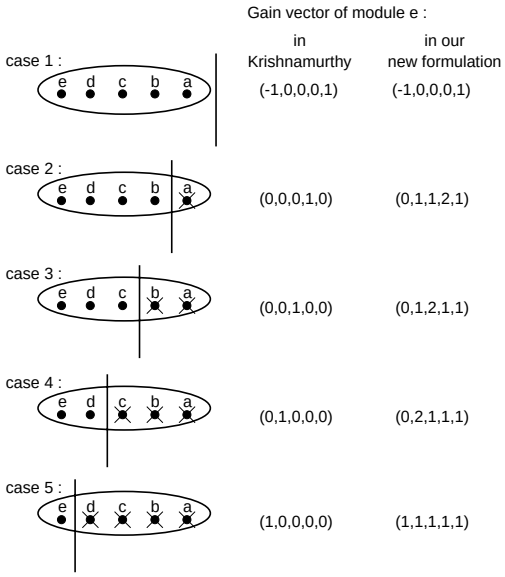

Figure 3: Evolution of the gain vector for module e according to the Krishnamurthy level gain formulation and our new gain formulation.

Experimental Results

We tested our new gain formulation using the same LIFO, random and FIFO selection schemes described in Section 2. The results are shown in Table 2. Note that the third column (pure FM) results are the same as in the third column of Table 1 since our new formulation does not affect the first-level gain. As was observed with the Krishnamurthy formulation, the results using a LIFO selection scheme with our new formulation are significantly better than the results using random or FIFO selection schemes. However, the second-level gain results (column 4) using random and FIFO selection schemes show significant improvement over the pure FM results (column 3) with our new formulation. This is in sharp contrast to the results using the Krishnamurthy formulation, which did not show much improvement with higher-level gains using either random or FIFO selection. It may be that our new formulation tends to compute higher-level gains more carefully, thus obviating the need for a "good" selection scheme (i.e., the results for random and FIFO will more closely mirror the results of LIFO as the length of the gain vectors increases). Also, our new formulation explicitly gives higher priority to the neighbors of moved modules, which is similar to the effect of the LIFO selection scheme.

Table 3 compares LIFO results using our new formulation against LIFO results using the original Krishnamurthy formulation. In some cases our formulation leads to substantial reduction in the size of the cuts found.

4 Conclusion

We have found that implementation choices play an important role for both the FM [2] and Krishnamurthy [5] algorithms. In particular, selection from gain buckets based on the implicit ordering of a linked list representation is highly advantageous, and results in improved partitioning solutions. We find that eliminating randomization from the bucket selection not only improves the solution quality, but has a greater impact on FM performance than adding the Krishnamurthy gain vector. Organizing the gain buckets as LIFO (Last-In-First-Out) stacks leads to a 35% improvement versus random bucket organization and a 42% improvement versus FIFO (First-In-First-Out) queues. We have also presented an alternative higher-level gain formulation, based on Krishnamurthy's approach, which incorporates some of the intuition behind the LIFO organization. This alternative formulation results in a further 16% reduction in the average cut cost when compared directly to the Krishnamurthy formulation for higher-level gains, assuming LIFO organization for the gain buckets.

We believe that a much more detailed study is necessary to better understand the effect of "obvious" choices in the FM implementation on the solution

Ckt (#nodes)	Method	# Krishnamurthy levels $k=1$	$k=2$	$k=3$	$k=4$
Prim1 (833)	LIFO	83	76	75	74
	RAND	108	78	76	78
	FIFO	125	81	78	77
struct (1952)	LIFO	68	65	59	61
	RAND	185	55	57	56
	FIFO	202	60	55	57
Prim2 (3014)	LIFO	296	261	257	259
	RAND	441	262	256	260
	FIFO	507	283	267	267
biomed (6514)	LIFO	172	157	156	165
	RAND	450	199	179	183
	FIFO	503	180	180	172
ind2 (12637)	LIFO	757	619	623	623
	RAND	1671	809	683	692
	FIFO	1766	786	722	668
ind3 (15433)	LIFO	755	703	722	711
	RAND	2382	740	707	680
	FIFO	3285	750	712	743
avq.sml (21918)	LIFO	835	631	653	658
	RAND	1507	786	799	780
	FIFO	1842	863	824	815
avq.lrg (25178)	LIFO	1140	684	752	700
	RAND	1837	964	1035	951
	FIFO	2193	1114	1135	1054
% Impr. vs. RAND	LIFO	48.3	10.4	7.7	6.6
	RAND	0	0	0	0
	FIFO	-17.1	-4.5	-2.9	-2.3

Table 2: Average cutsize results for 100 runs of our new multi-level gain formulation (columns 4-6) using LIFO, random, and FIFO organization schemes for the gain buckets.

Ckt (#nodes)	Method	# Krishnamurthy levels $k=1$	$k=2$	$k=3$	$k=4$
Prim1 (833)	Ours	83	76	75	74
	Krish.	83	80	79	80
struct (1952)	Ours	68	65	59	61
	Krish.	68	66	65	66
Prim2 (3014)	Ours	296	261	257	259
	Krish.	296	281	279	275
biomed (6514)	Ours	172	157	156	165
	Krish.	172	181	199	220
ind2 (12637)	Ours	757	619	623	623
	Krish.	757	744	775	760
ind3 (15433)	Ours	755	703	722	711
	Krish.	755	799	797	815
avq.sml (21918)	Ours	835	631	653	658
	Krish.	835	911	950	964
avq.lrg (25178)	Ours	1140	684	752	700
	Krish.	1140	971	1012	1030
% Impr. over Krish.		0	14.5	16.2	19.4

Table 3: Results comparing our new multi-level gain formulation with Krishnamurthy's multi-level gain formulation using LIFO organization for the gain buckets. Averages are based on 100 runs.

quality and runtime. Thus, our future work investigates not only further tie-breaking mechanisms, but also interesting effects that result from the order imposed by the netlist representation and the list of free modules.[5] Studies of the LIFO organization scheme in multi-way partitioning and in more sophisticated partitioning approaches such as the two-phase FM methodology are also under investigation.

References

[1] S. Dutt. New faster kernighan-lin-type graph-partitioning algorithms. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 370–377, 1993.

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

[3] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.*, 49(2):291–307, 1970.

[4] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[5] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Trans. on Computers*, 33(5):438–446, 1984.

[6] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley-Teubner, 1990.

[7] L. A. Sanchis. Multiple-way network partitioning. *IEEE Trans. on Computers*, 38:62–81, 1989.

[8] L. A. Sanchis, *personal communication*, March 1994.

[9] D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. In *Proc. ACM/IEEE Design Automation Conf.*, pages 57–62, 1972.

[10] C. Sechen. *Placement and Global Routing of Integrated Circuits Using Simulated Annealing.* PhD thesis, Univ. of California, Berkeley, 1986.

[11] Y.-C. Wei and C.-K. Cheng. Towards efficient hierarchical designs by ratio cut partitioning. In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pages 298–301, 1989.

[5] The input format of a netlist is typically a function of how the other development tools represent and output the circuit, and may group related nets or modules together or far apart. This relatedness/unrelatedness will in turn be reflected within the data structures used by FM to store the netlist information.