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 90095-1596
† Cadence Design Systems, Inc., San Jose, CA 95134

Abstract

Iterative improvement partitioning algorithms such as the FM algorithm of Fiduccia and Mattheyses [8], the algorithm of Krishnamurthy [13], and Sanchis's extensions of these algorithms to multi-way partitioning [16], all rely on efficient data structures to select the modules to be moved from one partition to the other. The implementation choices for one of these data structures, the *gain bucket*, is investigated. Surprisingly, selection from gain buckets maintained as LIFO (Last-In-First-Out) stacks leads to significantly better results than gain buckets maintained randomly (as in previous studies of the FM algorithm [13] [16]) or as FIFO (First-In-First-Out) queues. In particular, LIFO buckets result in a 36% improvement over random buckets and a 43% improvement over FIFO buckets for minimum-cut bisection. 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. The LIFO selection scheme also results in improvement over random schemes for multi-way partitioning [16] and for more sophisticated partitioning strategies such as the two-phase FM methodology [2]. Finally, by combining insights from the LIFO gain buckets with the Krishnamurthy higher-level gain formulation, a new higher-level gain formulation is proposed. This alternative formulation results in a further 22% 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 Introduction

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 partitioning algorithms are typically variants of the Kernighan-Lin method (KL) [11] [18] or its algorithmic speedup by Fiduccia and Mattheyses (FM) [8]. Examples of more recent methods are given by Krishnamurthy [13], Sanchis [16], Dutt [7], Hoffman [10], Dasdan and Aykanat [6], and Saab [15]. Iterative improvement algorithms such as KL and FM start with a current feasible solution and

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

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 (or "move") 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". In netlist partitioning, the cost function is typically the number of nets cut; a given move has *gain* corresponding to the decrease in cut nets that would result from the move.

Over the past decade, FM has become perhaps the single most widely used and cited partitioning algorithm in the VLSI CAD area. The primary difference between the KL and FM algorithms lies in their respective neighborhood operators. KL iteratively makes a highest-gain swap of a pair of modules between two partitions; FM iteratively makes a highest-gain shift of a single module from one partition to another.[1] This subtle difference allows FM to achieve 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, where p is the total number of pins. The enabling data structure is an array of "gain buckets" which groups the modules of a given partition according to their gains.

Many works have investigated possible improvements and extensions to the FM algorithm. One commonly-cited extension is that of Krishnamurthy [13], which introduces efficient "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 stores a sequence of potential gain values corresponding to sets of future moves. 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.

[1] Each algorithm generates a *pass* of such moves until every module has been moved exactly once, then adopts the prefix of this move sequence with highest total gain. When a pass results in zero gain, the algorithm terminates.

[2] The notation used for the Krishnamurthy formulae are adapted from [13]. Note that in order to handle 1-pin nets correctly, the term $\beta_U(s) > 0$ should 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.

Intuitively, the positive term (i.e., first term) in the formula for γ_k counts nets with binding number $k-1$ that are "created" (for the partition that the module is moving "from") by the move, while the negative term (i.e., second term) counts nets with binding number $k-1$ that are "destroyed" (for the partition that the module is moving "to") by the move. Krishnamurthy's method uses lexicographic ordering of the vectors $(\gamma_1, \gamma_2, \gamma_3, \ldots)$ 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 only $O(kp)$ added computational expense, where k is the number of values maintained in (i.e., the size of) the gain vector. This was confirmed by Sanchis [16], who extended FM+HL (and thus implicitly FM also) to multi-way partitioning.

2 Tie-Breaking in the Fiduccia-Mattheyses Algorithm

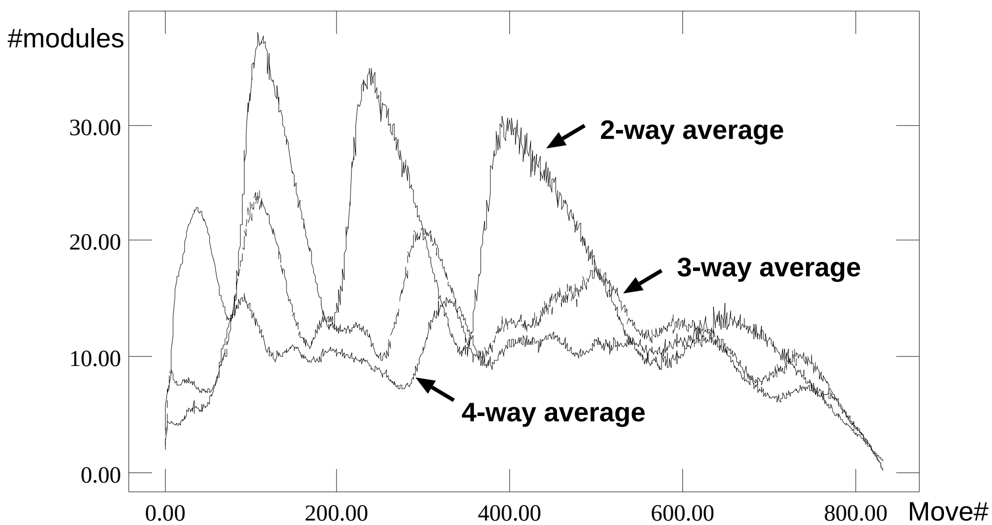

Figure 1: Average number of modules in the highest-gain bucket at each move during the first pass of FM on 2-way, 3-way and 4-way balanced partitioning for test case Primary1 (833 modules). The average numbers were generated from 1000 separate FM runs.

During a typical pass of FM, 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 in 2-way, 3-way and 4-way balanced partitioning (we plot the average over 1000 runs). Note that on average there are more modules in the highest-gain bucket during 2-way partitioning than during 3-way or 4-way partitioning.

From the figure accompanying the algorithm description in [8] one can infer that the Fiduccia and

Mattheyses gain buckets function as LIFO stacks (remove at head, insert at head). However, the gain buckets could just as easily function as FIFO queues (remove at head, insert at tail) while supporting the same algorithmic complexity. Neither Krishnamurthy nor Sanchis points out any tie-breaking schemes for cells with identical gains. In fact, both used randomized selection in case of tie.

In the following section we compare LIFO selection with random selection (used by Sanchis [16]) and FIFO selection (an alternative organization which as far as we know has never been used before). Our testbed is the code distributed by Sanchis [17] with appropriate modifications made for handling LIFO and FIFO selection. In all of our experiments, we assume the modules have unit area and constrain the partition sizes to differ by at most 1.

3 Experimental Results for Min Cut Bisection

Benchmark (#nodes)	Method	Pure FM Ave.(Min)	2nd-level Gain Ave.(Min)	3rd-level Gain Ave.(Min)	4th-level Gain Ave.(Min)
industry2 (12637)	LIFO	757.09(401)	743.57(453)	774.79(407)	760.47(439)
	RANDOM	1670.58(1142)	1299.16(834)	1108.28(665)	977.26(586)
	FIFO	1765.94(1432)	1342.30(1113)	1188.18(835)	1020.79(554)
industry3 (15433)	LIFO	755.27(315)	798.68(361)	796.81(325)	814.60(338)
	RANDOM	2381.63(1381)	1163.55(531)	1011.66(424)	952.48(344)
	FIFO	3285.25(1921)	1452.46(598)	1151.14(548)	1154.22(456)
avq.small (21918)	LIFO	834.92(542)	910.91(675)	950.31(577)	963.52(627)
	RANDOM	1507.03(1335)	1403.37(1277)	1383.65(1240)	1387.77(1223)
	FIFO	1841.80(1681)	1578.77(1441)	1498.67(1318)	1489.15(1279)
avq.large (25178)	LIFO	1140.22(816)	970.67(598)	1011.70(545)	1030.13(459)
	RANDOM	1836.89(1649)	1582.17(1416)	1527.13(1383)	1510.19(1328)
	FIFO	2192.99(2058)	1867.08(1734)	1767.62(1604)	1741.74(1609)
Average Improv. vs. RANDOM	LIFO	51.37%	36.97%	29.10%	24.75%
	RANDOM	0	0	0	0
	FIFO	-21.31%	-14.66%	-11.26%	-12.07%

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. The numbers in parentheses give the minimum cutsizes observed over 100 runs.

Our first experiment compares tie-breaking schemes in (2-way) minimum-cut bisection. Table 1 gives the average and minimum cutsizes for 100 FM runs using the three selection schemes (LIFO, random and FIFO). The "Pure FM" column of Table 1 clearly shows the effects of the selection methodology. Surprisingly, the FIFO scheme is much worse than random selection. The LIFO scheme, on the other hand, gives considerable improvement over random selection. A possible explanation for this improvement may be that organizing the buckets such that the "most recently visited" modules are placed near the beginning of the 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.[3]

Columns 4-6 of Table 1 show the effects of LIFO, random and FIFO selection on the Krishnamurthy higher-level gains [13]. Introducing second-level ($k = 2$) and in some cases third-level ($k = 3$) gain seems to improve the solution quality for random and FIFO selection. With regard to 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 gain elements considered.
- For test cases 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).

4 Experimental Results for Multi-Way Partitioning

We have also tested the LIFO, random and FIFO selection schemes for 3-way and 4-way balanced partitioning using Sanchis's [16] extension of the FM algorithm and Krishnamurthy's higher-level gains to multi-way partitioning. Tables 2 and 3 give the average and minimum cutsizes for 50 FM runs

[3] In bipartitioning, the cost at the end of the pass is exactly the same as the cost at the beginning of the pass.

Benchmark (#nodes)	Method	Pure FM Ave.(Min)	2nd-level Gain Ave.(Min)	3rd-level Gain Ave.(Min)	4th-level Gain Ave.(Min)
industry2 (12637)	LIFO	2316.12(2020)	2126.39(1873)	2103.59(1760)	2083.08(1517)
	RANDOM	2482.34(2308)	2182.02(1876)	2119.28(1761)	2124.90(1585)
	FIFO	2554.18(2346)	2159.87(1815)	2124.80(1739)	2136.78(1740)
industry3 (15433)	LIFO	3891.39(2718)	3592.73(2852)	3465.39(2624)	3449.68(2102)
	RANDOM	4393.32(4044)	3679.82(2896)	3584.32(2869)	3587.62(2789)
	FIFO	4574.50(3993)	3793.00(2859)	3567.18(2827)	3655.98(3103)
avq.small (21918)	LIFO	2634.36(2434)	2498.24(2374)	2499.12(2343)	2483.48(2331)
	RANDOM	2707.12(2562)	2572.88(2448)	2527.56(2390)	2516.36(2375)
	FIFO	2846.60(2659)	2595.74(2435)	2537.12(2416)	2536.82(2364)
avq.large (25178)	LIFO	3000.82(2876)	2721.34(2610)	2702.34(2599)	2709.92(2581)
	RANDOM	3034.54(2902)	2776.88(2651)	2733.76(2560)	2738.39(2616)
	FIFO	3132.34(2988)	2801.14(2703)	2742.04(2593)	2733.66(2618)
Average Improv. vs. RANDOM	LIFO	5.48%	2.45%	1.58%	2.04%
	RANDOM	0	0	0	0
	FIFO	-3.85%	-0.96%	-0.12%	-0.78%

Table 2: Average cutsize results for 50 runs of 3-way FM and Krishnamurthy higher-level gains (columns 4-6) using LIFO, random, and FIFO organization schemes for the gain buckets. The numbers in parentheses give the minimum observed cutsizes.

Benchmark (#nodes)	Method	Pure FM Ave.(Min)	2nd-level Gain Ave.(Min)	3rd-level Gain Ave.(Min)	4th-level Gain Ave.(Min)
industry2 (12637)	LIFO	2624.50(2346)	2441.78(2193)	2415.48(2191)	2387.90(1905)
	RANDOM	2778.54(2605)	2489.92(2254)	2441.16(2185)	2402.74(2047)
	FIFO	2829.68(2643)	2448.56(2294)	2415.08(2149)	2399.70(2136)
industry3 (15433)	LIFO	4968.88(4506)	4469.78(3575)	4366.54(3437)	4312.96(3594)
	RANDOM	5260.00(4885)	4497.02(3780)	4445.40(3676)	4311.02(3559)
	FIFO	5529.56(5196)	4575.36(4197)	4358.54(3590)	4352.42(3520)
avq.small (21918)	LIFO	3247.94(3055)	2921.18(2755)	2900.58(2734)	2889.34(2707)
	RANDOM	3443.82(3241)	2986.47(2889)	2920.42(2782)	2940.38(2817)
	FIFO	3597.83(3386)	3014.98(2877)	2924.66(2817)	2917.80(2776)
avq.large (25178)	LIFO	3573.78(3402)	3223.38(3050)	3154.78(3024)	3166.92(3070)
	RANDOM	3771.10(3614)	3261.08(3114)	3175.18(3042)	3167.34(3002)
	FIFO	3909.34(3660)	3298.88(3094)	3169.58(3012)	3130.94(3018)
Average Improv. vs. RANDOM	LIFO	5.50%	1.47%	1.04%	0.58%
	RANDOM	0	0	0	0
	FIFO	-3.78%	-0.55%	0.76%	0.27%

Table 3: Average cutsize results for 50 runs of 4-way FM and Krishnamurthy higher-level gains (columns 4-6) using LIFO, random, and FIFO organization schemes for the gain buckets. The numbers in parentheses give the minimum observed cutsizes.

using the three selection schemes (LIFO, random and FIFO). We measure cutsize as the number of nets cut by the partitioning. The results show that although LIFO gives better results than both random and FIFO, the overall gain is much less than was observed for 2-way partitioning. Furthermore, the Krishnamurthy higher-level gains appear to play a bigger part for multi-way partitioning than for 2-way partitioning.

Notice that there is much less variability in the multi-way partitioning results, i.e., the minimum is relatively close in value to the average. One possible explanation for the lack of solution variance might be that having on average fewer modules in the highest-gain bucket (recall Figure 1) may reduce the importance of tie-breaking. Another observation, which may help explain why the Krishnamurthy gain vector is more important for multi-way partitioning, is that there are now more than one possible destination partitions for a given module; thus, the direction in which a module may be "pulled" by previously moved modules is no longer unique.

5 Experimental Results for Two-Phase FM

	Avg cutsize (min cutsize)			Improvement of
Benchmark	LIFO	RANDOM	FIFO	LIFO vs. RANDOM
industry2	419.16(260)	453.69(301)	454.21(304)	8%
industry3	564.71(292)	630.72(327)	643.31(312)	10%
avq.small	408.79(206)	480.04(275)	499.11(297)	15%
avq.large	645.63(415)	745.35(495)	768.35(519)	13%

Table 4: Average cutsize results for 100 runs of two-phase FM for minimum-cut bisection using LIFO, random, and FIFO organization schemes for the gain buckets. The numbers in parentheses give the minimum observed cutsizes.

Our third set of experiments tests the LIFO, random and FIFO selection schemes within the so-called two-phase FM approach. Two-phase FM [3] [9], which has gained attention recently due to much better results than "single-phase" FM, is essentially the state-of-the-art in iterative partitioning (see, e.g., [5]). The method generates a partition by running two FM phases on the netlist: in phase I, a clustering of the netlist is constructed and FM is run on this clustered instance, after which phase II uses the "flattened" solution from phase I as the starting solution for running FM on the original netlist.[4] Table 4 gives the LIFO, random and FIFO results for 2-phase FM using the recent WINDOW clustering of [2].[5] As in Table 1, there is a noticeable difference among the three selection schemes. LIFO selection is consistently better than either random or FIFO selection, with average improvement of LIFO over random being 12%. Furthermore, the LIFO two-phase FM results are 41% better than the LIFO results for single-phase FM. These results demonstrate that the bucket organization also plays a part within sophisticated partitioning approaches.

[4]The balance constraint of the FM algorithm on the clustered netlist is set to half the total area $\pm$ the size of the largest cluster. This "relaxation" of the balance constraint is necessary to allow FM to find a good partitioning solution of the clustered netlist. In phase II, which uses the clustered netlist solution as the starting point, an initial set of greedy moves is performed to get a solution satisfying the balance constraints for bisection (half the total area ± 1).

[5]The WINDOW clustering algorithm first generates a linear ordering of the modules according to a specific "attraction function" and then uses dynamic programming to split the linear ordering optimally based on the clustering objective function. The orderings used correspond to the "scaled cost" (generalized ratio-cut) metric proposed by Chan et al. [4].

6 A Krishnamurthy Variant for Min Cut Bisection

The observation in Section 3—that it may be more important to move modules which are incident to locked nets—suggests an alternative multi-level gain formulation for 2-way partitioning. If a net is cut, and only one partition contains locked modules incident to this net, higher priority in moving should be given to the modules in the partition having no locked modules incident to this net. Such an objective can be achieved 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, for $k \geq 2$. We avoid changing the first-level gain since it should always reflect the "actual" gain resulting from a move of this module. However, we add 1 to all the other gain levels so that the increased priority is guaranteed to affect the tie-breaking.

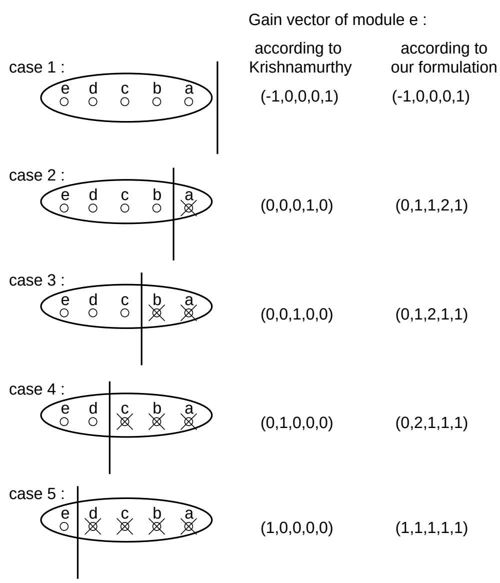

Figure 2: Evolution of the gain vector for module e according to the Krishnamurthy level gain formulation and our new gain formulation.

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

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

The first two terms are identical to Krishnamurthy's formulation [13]. The third term is new and represents the "attraction" to locked modules. Figure 2 contrasts the evolution of Krishnamurthy's gain vector against that of the gain vector resulting from our new formulation. Initially, 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 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 formulation, 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. Again, this may be an important difference since one would seemingly 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.

Benchmark (#nodes)	Method	Pure FM Ave.(Min)	2nd-level Gain Ave.(Min)	3rd-level Gain Ave.(Min)	4th-level Gain Ave.(Min)
industry2 (12637)	Ours	757.09(401)	618.82(334)	623.16(333)	622.62(275)
	Krishnamurthy	757.09(401)	743.57(453)	774.79(407)	760.47(439)
industry3 (15433)	Ours	755.27(315)	702.92(300)	721.96(349)	710.97(312)
	Krishnamurthy	755.27(315)	798.68(361)	796.81(325)	814.60(338)
avq.small (21918)	Ours	834.92(542)	630.81(398)	652.72(416)	657.84(373)
	Krishnamurthy	834.92(542)	910.91(675)	950.31(577)	963.52(627)
avq.large (25178)	Ours	1140.22(816)	683.85(457)	752.06(475)	699.59(406)
	Krishnamurthy	1140.22(816)	970.67(598)	1011.70(545)	1030.13(459)
Average Improvement		0	22.27%	21.49%	23.67%

Table 5: Results comparing our new multi-level gain formulation with Krishnamurthy's multi-level gain formulation using the LIFO organization schemes for the gain buckets. The averages are based on 100 runs; numbers in parentheses give the minimum observed cutsizes.

We tested our new gain formulation using the same LIFO, random and FIFO selection schemes described in Section 2. Table 5 compares LIFO results using our new formulation against LIFO results using the original Krishnamurthy formulation. The third column (pure FM) results are identical since our new formulation does not affect the first-level gain. Overall, we achieve 22% improvement over Krishnamurthy's formulation; in some cases (e.g., industry2 and avq.small) our formulation leads to substantial reduction in the size of the minimum cuts found.

Table 6 shows the LIFO, random and FIFO results for our new gain formulation. Just as 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. Notice, however, that the second-level gain results (column 4) using random and FIFO selection schemes show significant

Benchmark	Method	Pure FM	2nd-level Gain	3rd-level Gain	4th-level Gain
		Ave.(Min)	Ave.(Min)	Ave.(Min)	Ave.(Min)
industry2	LIFO	757.09(401)	618.82(334)	623.16(333)	622.62(275)
(12637)	RANDOM	1670.58(1142)	808.59(354)	682.66(327)	691.73(292)
	FIFO	1765.94(1432)	786.33(337)	722.07(324)	667.65(348)
industry3	LIFO	755.27(315)	702.92(300)	721.96(349)	710.97(312)
(15433)	RANDOM	2381.63(1381)	739.81(336)	706.63(307)	679.79(348)
	FIFO	3285.25(1921)	749.90(303)	712.04(328)	742.52(328)
avq.small	LIFO	834.92(542)	630.81(398)	652.72(416)	657.84(373)
(21918)	RANDOM	1507.03(1335)	785.81(641)	799.01(584)	780.41(591)
	FIFO	1841.80(1681)	862.82(663)	823.65(573)	814.61(634)
avq.large	LIFO	1140.22(816)	683.85(457)	752.06(475)	699.59(406)
(25178)	RANDOM	1836.89(1649)	964.04(769)	1034.63(848)	951.25(802)
	FIFO	2192.99(2058)	1114.02(935)	1135.28(884)	1053.74(840)
Average Improv.	LIFO	51.37%	19.31%	13.04%	11.89%
vs. RANDOM	RANDOM	0	0	0	0
	FIFO	-21.31%	-5.99%	-4.84%	-5.23%

Table 6: 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. The numbers in parentheses give the minimum observed cutsizes.

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. An explanation for this might be that with our new formulation, the higher-level gains are computed more carefully and tend to obviate 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.

7 Conclusion

We have shown that implementation choices play an important part for both the FM algorithm of Fiduccia and Mattheyses [8] and the algorithm of Krishnamurthy [13]. In particular, selection from gain buckets based on the implicit ordering of a linked list representation is advantageous, and will result in improved partitioning solutions. 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; this reopens the question of interpreting such seminal works in the literature as [16] and [13], whose studies used random bucket selection. Organizing the gain buckets as LIFO (Last-In-First-Out) stacks leads to a 36% improvement versus random bucket organization and a 43% 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 22% 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 more detailed study is necessary to better understand the effect of choices in the FM implementation on the solution 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.[6]

References

[1] C. J. Alpert and A. B. Kahng, "Geometric Embeddings for Faster and Better Multi-way Netlist Partitioning", Proc. ACM/IEEE Design Automation Conference, 1993, pp. 743-748.

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

[3] T. Bui, C. Heigham, C. Jones, and T. Leighton. "Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms." In *Proc. ACM/IEEE Design Automation Conf.*, pp. 775–778, 1989.

[4] P. K. Chan, M. D. F. Schlag, and J. Y. Zien. "Spectral K-Way Ratio-Cut Partitioning and Clustering." *IEEE Trans. Computer-Aided Design*, **13**(9):1088–1096, 1994.

[5] C.-K. Cheng and Y.-C. A. Wei. "An Improved Two-Way Partitioning Algorithm with Stable Performance." *IEEE Trans. Computer-Aided Design*, **10**(12):1502–1511, 1991.

[6] A. Dasdan and C. Aykanat. Improved multiple-way circuit partitioning algorithms. In *Proc. ACM/SIGDA International Workshop on Field-Programmable Gate Arrays*, 1994.

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

[6] 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.

[8] 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.

[9] L. Hagen and A. B. Kahng. "A New Approach to Effective Circuit Clustering." In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pp. 422–427, 1992.

[10] A. G. Hoffman. The dynamic locking heuristic – a new graph partitioning algorithm. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 173–176, 1994.

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

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

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

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

[15] Y. Saab. A fast and robust network bisection algorithm. *IEEE Trans. Computers (to appear)*, 1995.

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

[17] L. A. Sanchis, March 1994. Personal communication.

[18] 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.

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

[20] 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.