Combining Problem Reduction and Adaptive Multistart: A New Technique for Superior Iterative Partitioning

Lars W. Hagen, *Member, IEEE*, and Andrew B. Kahng

***Abstract*— VLSI netlist partitioning has been addressed chiefly by iterative methods (e.g., Kernighan–Lin and Fiduccia–Mattheyses) and spectral methods (e.g., Hagen–Kahng). Iterative methods are the de facto industry standard, but suffer diminished stability and solution quality when instances grow large. Spectral methods have achieved high-quality solutions, particularly for the ratio cut objective, but suffer excessive memory requirements and the inability to capture practical constraints (preplacements, variable module areas, etc.). This work develops a new approach to Fiduccia–Mattheyses (FM)-based iterative partitioning. We combine two concepts: 1) *problem reduction* using clustering and the two-phase FM methodology and 2) *adaptive multistart*, i.e., the intelligent selection of starting points for the iterative optimization, based on the results of previous optimizations. The resulting *clustered adaptive multistart* (CAMS) methodology [18] substantially improves upon previous partitioning results in the literature, for both unit module areas and actual module areas, and for both the min-cut bisection and minimum ratio cut objectives. The CAMS method is surprisingly fast and has very stable solution quality, even for large benchmark instances. It has been applied as the basis of a clustering methodology within an industry placement tool.**

I. Introduction

PARTITIONING optimizations are critical to the synthesis of large-scale VLSI systems. Designs with over a million transistors are now quite common, and entail problem complexities that are unmanageable for existing back-end physical layout tools. Thus, partitioning is used to divide the design into smaller, more manageable components. With system design being increasingly dominated by performance and I/O constraints, the traditional goal of partitioning has been to minimize the number of signals which pass between components.

A. Partitioning Methods

A standard model for VLSI layout associates a hypergraph $G = (V, E)$ with the circuit netlist; vertices in V represent modules, and hyperedges in E represent signal nets. A bipartitioning of G divides the vertices in V into disjoint subsets U and W, with the *cut size* $c(U, W)$ being the number of hyperedges in $\{e \in E | \exists u, w \in e$ with $u \in U$ and $w \in W\}$. Two basic partitioning formulations are: 1) *min-width bisection*, which seeks the partition with $|U| = |W|$ such that $c(U, W)$ is minimized, and 2) *minimum ratio cut*, which seeks the partition such that $(c(U, W)/|U| \cdot |W|)$ is minimized. Both of these formulations are NP-complete, and much work has sought effective heuristic solutions.

Current partitioning approaches can be classified into *iterative* methods and *spectral* methods. Iterative methods are more widely used, and involve local perturbation of a current solution with either a greedy or a hill-climbing strategy. The iterative algorithm of Fiduccia and Mattheyses (FM) [13] (a variant of [21] that uses linear time per pass) is the method that is most widely used for bisection [23]. Wei and Cheng [30] use an adaptation of [13] to address the ratio cut objective. Spectral methods use eigenvectors of the Laplacian of a netlist-derived graph to deterministically find a partitioning solution. The determinism of spectral methods is appealing, and the need for only one run keeps CPU requirements reasonable as instances grow large. Spectral heuristics developed by Hagen and Kahng [15] use eigenvectors to define linear orderings of either modules or nets, and find good partitions by splitting the linear ordering. An extension in [10] produced results for ratio cut partitioning corresponding to an average of 28.8% improvement over the method of [30].

The main weakness of FM is that its solution quality is not very "stable," i.e., it is not predictable. Fig. 1 shows the FM solution cost distribution for the Primary2 benchmark netlist which has 3014 modules. The distribution is "normal," whereby the average FM solution is significantly worse than the best FM solution. Thus, FM must be run many times from random starting points to achieve a good result, i.e., to hit the tail of this distribution of solution costs. Indeed, practical implementations of FM use a number of random starting configurations and return the best result [23], [30] in order to attain "stability": we call this the *random multistart* approach. The number of runs required to achieve stability via random multistart grows very rapidly with problem size [19], [31].

Despite these shortfalls, iterative algorithms—and FM in particular—still possess many appealing advantages over the spectral approach. These advantages include smaller memory requirements, simplicity, and the ability to handle constraints such as preplacements or variable module areas. Thus, the

Manuscript received August 30, 1993; revised July 14, 1997. This paper was recommended by Associate Editor G. Zimmermann.

L. W. Hagen is with Cadence Design Systems, Inc., San Jose, CA 95134 USA.

A. B. Kahng is with Cadence Design Systems, Inc., San Jose, CA 95134 USA, on leave from the Department of Computer Science, University of California at Los Angeles, Los Angeles, CA 90095-1596 USA.

Publisher Item Identifier S 0278-0070(97)07769-5.

0278–0070/97$10.00 © 1997 IEEE

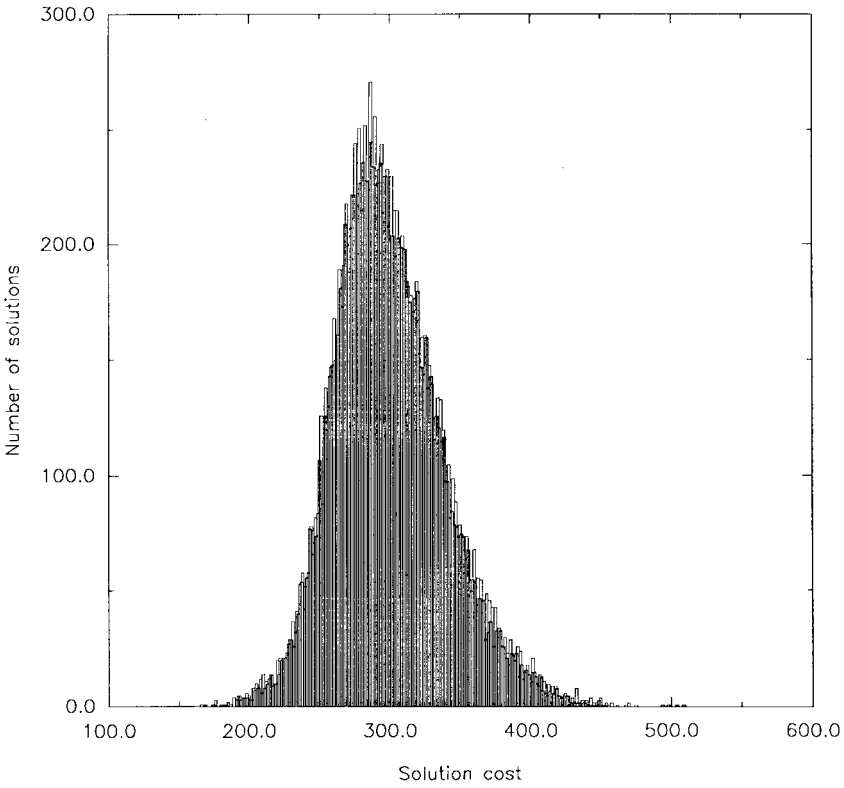


Fig. 1. Distribution of 21 203 FM bisection solutions (i.e., local minima) for SIGDA Layout Synthesis benchmark Primary2 (3014 modules). Each solution was generated from a new random starting point.

iterative FM approach retains a very strong appeal to practitioners, and much work has attempted to make FM more viable in practice.

II. Improvements to Iterative Partitioning

Two main approaches have improved the quality of multistart iterative partitioning:

1) reducing the problem size so that a smaller, more easily solved problem instance is obtained (this is the *clustering* or *two-phase FM* approach); and
2) intelligently constructing new starting points for the optimization based on previously found local minima (this is the *adaptive* multistart approach).

A. Clustering: Reducing the Problem Size

Informally, a *clustering* groups the netlist modules into disjoint subsets or *clusters*. Contracting the modules of each cluster into a single node induces a *compacted*, or *condensed*, representation of the original problem which may be easier to solve. Bui *et al.* [5], [6] proposed the "matching-based compaction" (MBC) algorithm, where the edges of a maximal random matching in the netlist graph induce a compacted instance of $\lceil n/2 \rceil$ vertices, and the compaction is iterated until the problem size becomes manageable. A *two-phase FM* methodology results: the FM algorithm is applied to the compacted netlist, and the result is reexpanded into a flat initial configuration for a second FM application. Ng *et al.* [26] used a similar approach with a clustering algorithm which attempted to minimize the Rent parameter of the condensed netlist; Ding *et al.* [12] also used Rent-based clustering to improve the performance of a placement algorithm.

Other two-phase FM approaches include [16], whose probabilistic RW-ST method finds "natural clusters" via a self-tuning *random walk* in the circuit netlist (strongly connected regions of the netlist are detected by multiple revisitations of modules within the walk). Cong and Smith [11] generalize the matching of Bui *et al.* to a clique-finding scheme, and provide a parallel implementation. The strongest two-phase FM results for netlist bisection seem to be those of Alpert and Kahng [2], whose AGG algorithm applies geometric clustering to a multidimensional spectral embedding of the netlist. Using AGG clusters in the two-phase FM approach yields bisections that are an average of 26.9% better than the results of running "flat" FM 200 times. Another strong result is reported by Cheng and Wei [8] for partitioning with a 1:3 size ratio bound (as opposed to exact bisection); their stable two-way (STW) partitioning algorithm uses recursive FM-based ratio cut partitioning [30] to achieve a circuit clustering, then applies the two-phase FM methodology.

B. Structure in the Solution Space: Adaptive Multistart (AMS)

The second approach to improving iterative methods has centered on the careful choice of an initial configuration for each execution of the optimization algorithm. In such an approach, previously generated solutions are used to construct a starting point that is more likely to lead to a good local minimum. Boese *et al.* [3] showed that the set of local

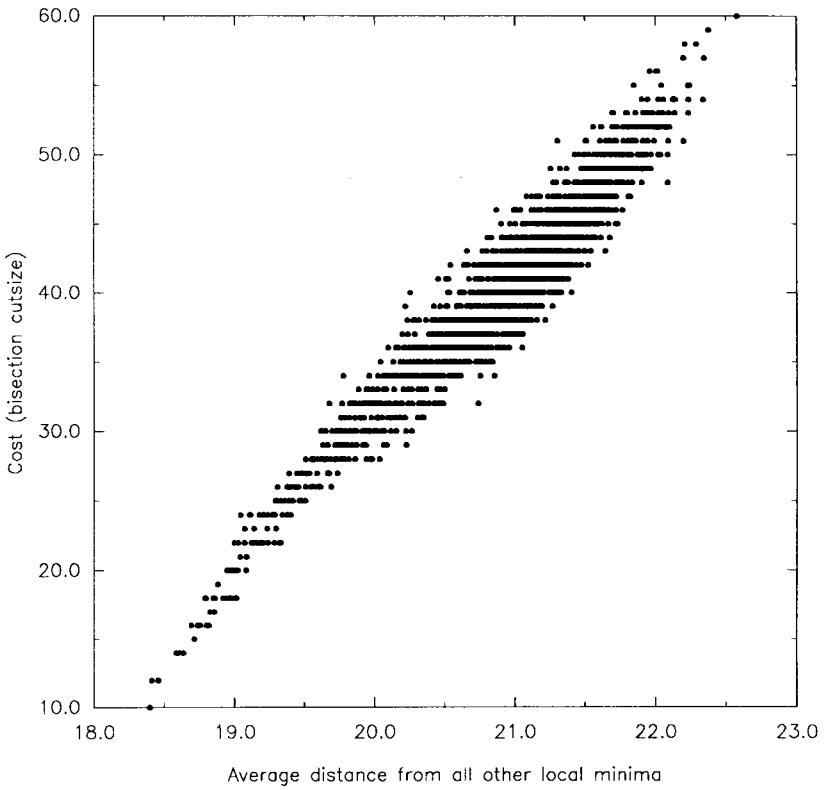


Fig. 2. Analysis of 2500 random local minimum bisections for graph in the class of difficult instances $G_{\mathrm{Bui}}(100, 4, 10)$. The number of unique local minima plotted is 2343. For each solution, we plot its cost against its average distance, in terms of single-vertex moves "shift moves," to all 2499 other solutions.

minima for many iterative algorithms, *under the appropriate neighborhood structure*, exhibits a "big valley." For a 100-node instance, Fig. 2 portrays 2500 local minimum graph bisections according to the pair-swap neighborhood structure.[1] Each solution is plotted by its cost (number of edges cut) versus its average distance (number of pair swaps) from all other solutions. There is a clear correlation: the best local minima are central to all of the others. Based on this observation, the *adaptive multistart* (AMS) methodology [3] consists of two phases.

1) Generate a set of *random* starting points, and call the iterative algorithm *Iter-Alg* on each starting point, thus determining a set of (local minimum) solutions.
2) Construct *adaptive* starting points from the best local minimum solutions found so far, and run *Iter-Alg* on these to yield corresponding new solutions.

Essentially, 2) attempts to find starting points that are central to the previous solutions, and thus are more likely to reach the center of the big valley.

Work in the genetic algorithms literature has discussed similar ideas within the context of "hybrid genetic-local search" [24] or "learn as you search" [1]. By combining genetic algorithms with local search strategies, [4], [25], [24], [29] showed that improved results were possible for the traveling salesman problem (TSP) and partitioning. The basic approach in these works allows an iterative algorithm to improve each individual, either before or while being combined with other individuals to form new solution "offspring." Such works mostly remain within the genetic paradigm in that new solutions are derived from only two "parents"; however, Mühlenbein [24] and Ackley [1, p. 35] describe multiparent, voting approaches for forming new solutions.

[1]This particular instance is from the class of "difficult" bisection inputs proposed by Bui *et al.* [5]. Specifically, a random graph in the class $G_{\mathrm{Bui}}(n, d, b)$ has n nodes, is d-regular, and has an optimum bisection cost almost certainly equal to b.

III. Clustered Adaptive Multistart (CAMS)

Clustering and AMS each improve on the naive (random multistart) implementation of iterative search. However, each has drawbacks.

- Clustering the netlist becomes expensive as instances grow large, and can constrain the two-phase FM approach to a solution space that does not include any high-quality solutions.
- AMS relies on the local minima of the iterative strategy exhibiting a "big valley" structure; however, Fig. 3 shows that with FM local minima, picking a central point will not obviously lead to a better solution.

Our contribution lies in combining the clustering and AMS philosophies into a new *clustered adaptive multistart* (CAMS) methodology [18] which enables the FM algorithm to rapidly return best known partitioning solutions. As with AMS, CAMS adaptively exploits previous good solutions, except that the solutions are used to find a *clustering*, and not a "central" solution. As with two-phase FM, the clustered input is then used as the input to FM, and the flattened result is the input to a second FM phase. Our intuitions are: 1) that CAMS

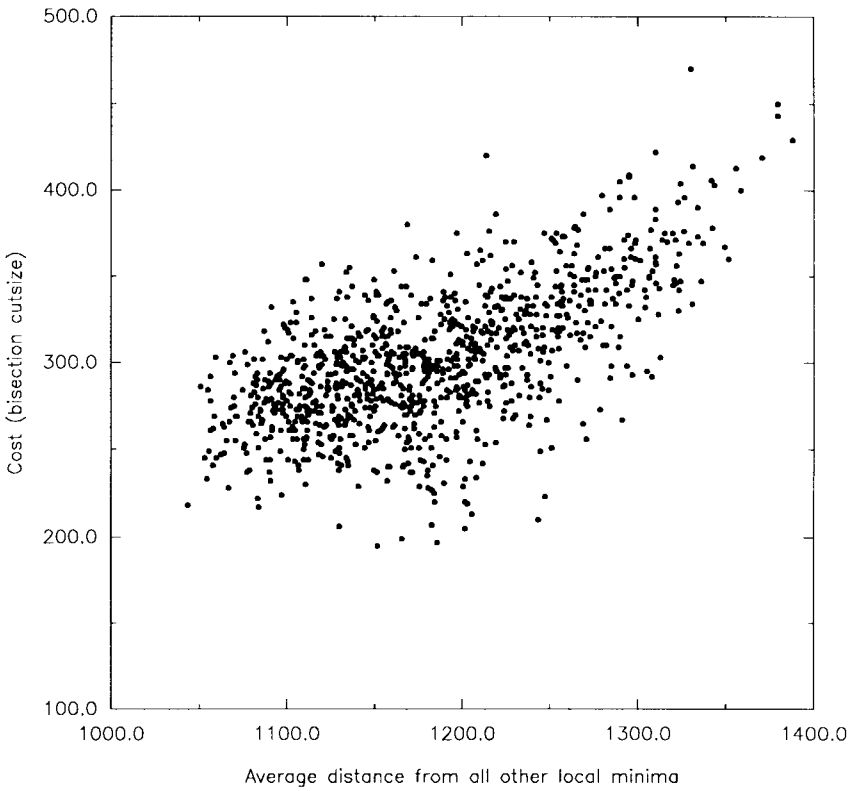


Fig. 3. Analysis of local minimum FM bisections of SIGDA Layout Synthesis benchmark Primary2, using 1000 random starting configurations. All 1000 local minima are distinct. For each solution, we plot its cost against its average distance, in terms of single-vertex moves "shift-moves," to all 999 other solutions.

utilizes the information stored in "good" local minima to predispose certain nodes toward specific partitions, and 2) that CAMS creates an "easier" problem by reducing the original netlist hypergraph to a smaller and more manageable clustered hypergraph.

The CAMS methodology is described in Fig. 4. A single pass of CAMS corresponds to the main loop, **Clustered_Adaptive_Multi-Start** (lines 5–18), where CAMS performs two-phase FM to generate k new solutions. The clustered hypergraph G' used within the two-phase FM runs is produced by the **Construct_Clustered_Netlist** subroutine from k previous solutions. After each pass through the main loop, the k previous solutions used to generate the clustering are replaced by the k new solutions generated during the pass.[2] In practice, the CAMS clusterings will, after several passes, almost always converge to a two-way partitioning (i.e., a clustering with only two clusters). While this in itself could be considered a natural stopping condition, we use the condition that the main loop will be exited if the best solution quality has not improved over the last two passes (the variable u in Fig. 4 counts the number of passes with no improvement). This stopping criterion can save a substantial amount of runtime since, for many instances, there will be a large number of passes with no improvement in solution quality as the CAMS clustering "converges" into a two-way partitioning.

[2]We have tested an alternate methodology where the clustered hypergraph is constructed from the k best solutions taken from over the entire history of the solution process. However, this methodology gave inferior average solution qualities compared to the methodology which constructs the clustered hypergraph from the k solutions found in the latest pass.

The subroutine **Construct_Clustered_Netlist** constructs the clustered hypergraph G' from k previous partitioning solutions of the hypergraph G: vertices of G that occur together in the same partition in all of the k solutions are grouped into a single condensed vertex in G'. In order to do this efficiently, each vertex in G is given a k-bit label, one bit from each partitioning solution, with each bit being 0 or 1 based on whether it is in the same partition or the opposite partition as vertex v_1. Finding the clusters is then simply a matter of identifying vertices which have identical labels: this is a bucket or radix sort on the vertex labels, and is accomplished in time linear in the number of vertices.[3] Fig. 5 shows an example of how **Construct_Clustered_Netlist** would construct G' from four two-way partitioning solutions.

We believe that the subroutine **Construct_Clustered_Netlist** provides the key to the success of the CAMS algorithm. Previous two-phase FM approaches build their clusters through hierarchical (bottom up in [5], top down in [8]) or randomized [16] processing of the *netlist*, and can take a "wrong turn" in the process. CAMS, on the other hand, through calls to **Construct_Clustered_Netlist**, extracts structural building blocks directly from the *actual partitioning solutions*. This idea of constructing the clusters from previous solutions, rather than the input instance, is quite novel; it significantly improves the solutions that can be generated by the two-phase FM approach.

While CAMS is similar to AMS in that many local minima are used to generate a promising starting point, a fundamental

[3]Asymptotically, the clustering operation is dominated by the complexity of the k FM executions which *per pass* require time linear in the size of the netlist.

Clustered_Adaptive_Multi-Start (G,D,k)

Input: Netlist hypergraph $G = (V, E)$ with $|V| = n$
$k \equiv$ number of solutions used to construct each clustering
Output: $t^* \equiv$ best solution found
Local Variables:
$M_1 \equiv$ set of k solutions
$M_2 \equiv$ set of k solutions
$u \equiv$ number of passes without improvement

```
1.  M1 ← k solutions from random starting points
2.  M2 ← ∅
3.  t* ← best solution in M1
4.  u ← 0
5.  repeat
6.      G' ← Construct_Clustered_Netlist(G,M1)
7.      for i = 1 to k
8.          t' ← Random_Partition(G')
9.          t' ← FM(G',t')
10.         t ← FM(G,Flatten(t'))
11.         M2 ← M2 ∪ t
12.     u ← u + 1
13.     t ← best solution in M2
14.     if Cost(t) < Cost(t*)
15.         t* ← t
16.         u ← 0
17.     M1 ← M2
18. until u = 2
```

(a)

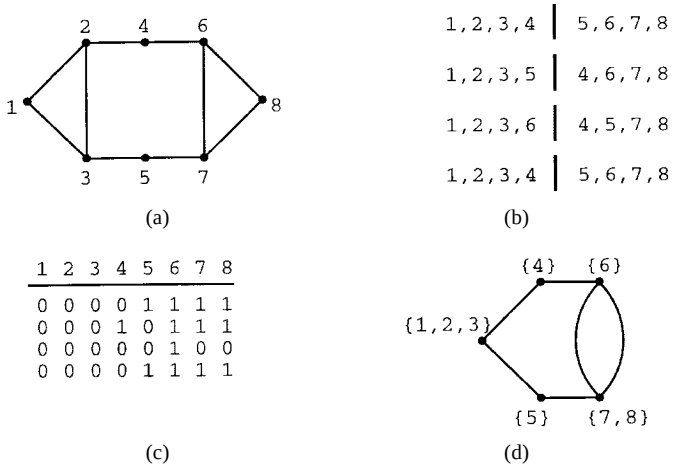
Subroutine Construct_Clustered_Netlist(G,M)

Input: Hypergraph $G = (V, E)$ with $|V| = n$
Set $M = \{M_1, \ldots, M_k\}$ of partitioning solutions
Output: A clustered hypergraph $G' = (V', E')$

S1. **for each** module $v_i \in V = \{v_1, \ldots, v_n\}$ **do**
S2. Construct k-bit label $L = L_1 \ldots L_k$
where $L_j \leftarrow 0$ if v_i and v_1 are in the same partition of M_j
and $L_j \leftarrow 1$ otherwise
S3. $V' \leftarrow$ the set of all labels created in Steps S1 and S2.
S4. $E' \leftarrow$ the set of hyperedges over V' that is induced by E

(b)

Fig. 4. **Clustered_Adaptive_Multi-Start** template.



(a) (b) (c) (d)

Fig. 5. Clustered graph constructed from four partitioning solutions using the subroutine **Construct_Clustered_Netlist**. The construction for hypergraphs is similar. (a) Original Graph. (b) Partitioning Solutions. (c) Bit-Vectors. (d) Clustered Graph.

difference between CAMS and AMS lies in their respective assumptions concerning the structure of local minima in the solution space. AMS explicitly depends on a "big valley" via the assumption that the centroid of local minima will lead to a good solution. CAMS is much more flexible: it assumes only that local minima have certain subsets in common, and that the problem size can be reduced by clustering these subsets; this seems to enable CAMS to spend most of its computation on the "hard" parts of the problem.[4]

IV. Experimental Results

We have performed the following experiments to compare CAMS against the previous work in partitioning:

1) comparison of CAMS against AMS and random multistart;
2) comparison of CAMS against the previous two-phase FM partitioning results in [2];
3) comparison of CAMS against the method of Cheng and Wei [8]; and
4) comparison of a ratio cut implementation of CAMS against previous ratio cut partitioning results in [30], [15], [10].

Our first experiments compared CAMS against AMS and random multistart using FM as the underlying optimization.[5]. Fig. 6 shows the performance of the three approaches for unit-area FM bisection on the SIGDA Layout Synthesis benchmark Primary2. All of the numbers shown are the averages of at least 50 independent runs, e.g., to get the single data point for 1000 random multistart calls to FM, it was necessary to call FM a total of 50 000 times with random starting points. The data for CAMS were found by varying k in the CAMS algorithm, i.e., the average number of FM calls used by CAMS is a function of the parameter k. We ran CAMS 50 times for different values of k, and used the resulting average solution quality and average number of FM calls as our data.[6] We varied k from $\lceil((1/2)\cdot\log_2 n)\rceil$ to $\lceil(2\cdot\log_2 n)\rceil$, where $n = |V|$ is the number of modules. Our reasoning was that if the k partitions differ maximally from each other, we require $k = \lceil\log_2 n\rceil$ to get no clustering at all. Hence, we used $k = \lceil\log_2 n\rceil$ as

[4] Recall that the correlations for FM local minima in Fig. 3 were much weaker than for pair-swap local minima in Fig. 2. Thus, the validity of the AMS paradigm is heavily dependent on the specific neighborhood structure that is used. Indeed, the results for AMS versus random multistart were much less spectacular with FM partitioning than with the other formulations/optimizations reported in [3]. See Fig. 6.

[5] Our version of FM is randomized so that more than one result is possible for a given starting point. This entailed some minor changes to the code, but did not produce any noticeable change in solution quality. Also note that our FM implementation does not take advantage of the enhancements proposed by Krishnamurthy [22]. The AMS results were derived from an implementation which follows the description in [3]. Our AMS implementation generates each new starting point stochastically, based on the k best known local minima. These k local minimum bisections are each given the "parity" such that their division of the modules into partitions 0 and 1 minimizes the move distance from the best local minimum bisection. (In other words, for a given bisection $V = (U, W)$, we will make U the "0 partition" and W the "1 partition" or vice versa, so that the bisection resembles the best known bisection as much as possible.) The modules are then assigned probabilities of membership in partitions 0 and 1 of the new starting point, based on their locations in the k solutions and according to a weighting function derived from the k solution costs. The weight contributed by each local minimum bisection is the cost of the bisection divided by the sum of the costs of all k bisections. After all module probabilities have been determined, each module is randomly assigned to either partition 0 or partition 1 according to these probabilities.

[6] In counting the number of FM calls CAMS makes, we are not counting the FM calls on the clustered netlist, i.e., one might argue that the number of FM calls is nearly twice that listed. This is the method of counting used in previous two-phase FM work [2]. Note that FM calls on the clustered netlist are of much lower complexity than those on the flattened netlist.

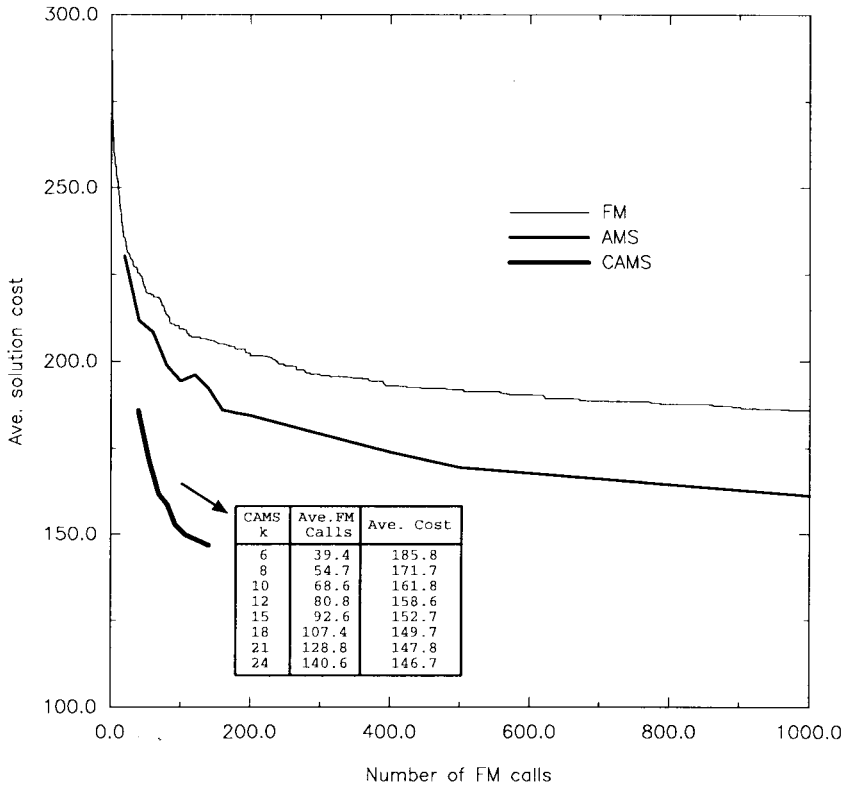


Fig. 6. Plot of average bisection solution quality as a function of total number of FM runs using SIGDA Layout Synthesis benchmark Primary2 with each module having unit area. The AMS plot is generated from the results for 20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 500, and 1000 FM calls. The CAMS plot is generated by varying the parameter k as shown in the table. Each data point corresponds to the average of at least 50 independent runs.

the midpoint of our range. In Fig. 6, we see that a CAMS run that required an average of 80.8 FM calls resulted in an average solution quality that would require more than 1000 FM calls in an AMS paradigm. We also notice that the improvement in average solution quality for the CAMS runs seems to taper off at around 110 FM calls,[7] which corresponds to $k \approx 18$. These and similar experiments prompted our choice of $k = \lceil (1.5 \cdot \log_2 n) \rceil$, and the remaining experiments reflect this choice.

A similar experiment was performed on the Industry2 benchmark (12 142 modules) for which there are no previously published partitioning results. The CAMS results for Industry2 give an average bisection cutsize of 211.72 using an average of 128.36 FM calls. The best solution we found using more than 8000 random multistart FM calls had cutsize 303, again indicating that CAMS provides a substantial improvement over the original multistart approach. The best solution found by our 50 runs of CAMS had cutsize 181.

Our second set of experiments tested CAMS against the previous two-phase FM partitioning results mentioned in Section II-A. Table I shows the average solution and best solution of 50 independently started CAMS runs. In these experiments, we use actual module area to conform with the previous two-phase FM results published in [2].

Our third set of experiments compared CAMS against the method of Cheng and Wei [8], which applies two-phase FM using clusters obtained through recursive ratio cut partitioning. We adopt the same experimental methodology as in [8], i.e., we report net cuts subject to a 1:3 size ratio bound, using actual module areas. It is very important to note that the results in Table II can be *worse* than the corresponding bisection results in Table I. This is because the FM *bisection* allows imbalance of up to the largest module area during the shift/swap procedure—and for such examples as Test04, this is greater than 25% of the total module area, i.e., the tolerance applied for the 1:3 ratio-bounded optimization. For each benchmark, CAMS again finds a partition of either equal

TABLE I
COMPARISON OF BISECTION SOLUTION QUALITY USING ACTUAL MODULE AREAS ON BENCHMARKS FROM THE SIGDA LAYOUT SYNTHESIS SUITE

Benchmark	Best Previous	CAMS Best	CAMS Average	Runtime (s)
bm1	48 (AGG)	47	50.76	118.61
19ks	124 (AGG)	110	114.08	1159.76
PrimGA1	47 (RW-ST)	47	50.72	112.52
PrimGA2	146 (AGG)	138	147.10	826.43
PrimSC1	49 (AGG)	47	51.32	111.61
PrimSC2	144 (AGG)	135	142.28	833.72
Test02	42 (All)	42	42.02	295.54
Test03	50 (AGG)	39	39.16	263.90
Test04	12 (FM&AGG)	12	12.62	387.10
Test05	24 (FM)	24	35.32	773.79
Test06	63 (AGG)	63	64.06	464.65

The "Best Previous" column contains the best previously published result, and in parentheses gives the algorithm which generated that result. Note that the Test05 bisections in [16] and [2] used slightly corrupted module area data, and therefore reported incorrect bisection costs. The entries in this table reflect our rerunning RW-ST and AGG on the corrected input file for Test05.

[7] The best net cut observed over all the runs performed was 146. Since the average solution quality is 146.7 when $k = 24$, the plot for CAMS will have a slope very near zero for larger values of k.

TABLE II
COMPARISON WITH METHOD OF CHENG AND WEI [8] FOR MINIMUM NET CUT VALUE SUBJECT TO 1:3 PARTITION SIZE RATIO BOUND

	Cheng-Wei 91				CAMS			
Benchmark	Best	Avg.	σ	Runtime (s)	Best	Avg.	σ	Runtime (s)
bm1	17	17.95	1.27	33.85	17	17.70	1.62	71.40
19ks	80	91.00	5.85	228.80	72	77.10	2.43	756.23
PrimGA1	17	17.25	0.91	34.25	17	17.80	2.02	65.04
PrimGA2	77	77.70	2.68	128.30	77	79.32	6.24	578.73
PrimSC1	17	18.20	0.68	29.70	17	17.68	1.88	67.18
PrimSC2	77	77.90	2.84	176.35	77	77.82	2.65	570.49
Test02	42	42.00	0.00	138.70	42	42.50	1.50	276.99
Test03	39	39.55	1.50	66.45	39	39.16	0.78	244.64
Test04	42	43.40	1.05	67.65	42	43.88	0.48	186.93
Test05	42	42.20	0.62	156.50	42	42.00	0.00	466.04
Test06	55	60.85	3.76	142.25	50	50.24	0.81	343.42
26K	65	156.95	64.03	4767.05	49	58.00	17.30	8229.82

To achieve an exact comparison, we report minimum, mean, and standard deviation for the net cut value. Results of Cheng and Wei are based on 20 runs, while the CAMS results are based on 50 runs. Note that the $1:3$ size ratio bound can entail a *tighter* constraint on the FM optimization than the bisection optimization discussed in Table I.

or better solution quality when compared to the previous work. Indeed, for the benchmarks 19 ks, Test06, and 26 K, the *average* CAMS solution quality is substantially better than the best solution reported in [8]. The results for benchmark 26 K, which contains almost 26 000 modules, are especially noteworthy since they may indicate the future success of CAMS as instances become larger.

Our final set of experiments tested CAMS against previous *ratio cut* partitioning results. Table III shows the average and best solution quality for 50 independent CAMS runs which optimize the ratio cut objective; these results are compared against the best ratio cut results in the literature (quoted for RCut1.0 [30], EIG1 [15], EIG1-IG [15], and IG-Match [10]—the last three are spectral methods). Here, we use unit module areas, again to maintain comparability against the previous results. For these experiments, we used a version of the FM algorithm that was slightly modified so as to minimize the ratio cut objective instead of the bisection objective.

For many of the benchmarks, the CAMS results are not that much better than the spectral results (of course, the spectral results already average almost 30% better than FM-based ratio cuts). This supports the claim in [15] that spectral methods yield high-quality ratio cut partitions. However, as noted above, even though the solution qualities are comparable for unit-area instances, FM-based methods are greatly preferable for their inherent robustness and amenability to practical constraints such as variable module areas.

TABLE III
COMPARISON OF RATIO CUT SOLUTION QUALITY ON BENCHMARKS FROM THE SIGDA LAYOUT SYNTHESIS SUITE WITH ALL MODULES HAVING UNIT AREA

Benchmark	Best Previous	CAMS Best	CAMS Average	Runtime (s)
bm1	5.531e-05 (EIG1, IG)	5.531e-05	6.068e-05	83.81
19ks	5.882e-05 (RCut1.0)	4.394e-05	4.547e-05	1078.74
Prim1	1.339e-04 (IG)	1.339e-04	1.384e-04	91.11
Prim2	4.576e-05 (IG-Match)	4.576e-05	4.612e-05	901.34
Test02	1.240e-04 (IG-Match)	7.962e-05	9.521e-05	518.26
Test03	8.984e-05 (IG-Match)	8.612e-05	9.045e-05	349.81
Test04	5.700e-05 (IG-Match)	5.700e-05	5.735e-05	404.78
Test05	3.060e-05 (IG-Match)	3.060e-05	3.249e-05	1192.36
Test06	7.484e-05 (IG-Match)	7.738e-05	7.867e-05	504.22

The "Best Previous" column contains the best previously published result, and indicates in parentheses the algorithm which generated this result. IG indicates that both EIG1-IG and IG-Match obtained the same result. The CAMS results are based on 50 runs.

V. DISCUSSION AND CONCLUSIONS

The success of the clustered adaptive multistart algorithm provides new insights into previous works and beliefs in the literature. In particular, while CAMS was originally formulated to draw on the established techniques of clustering and adaptive multistart, we find that quite a bit of the original intuition behind these two approaches is now subject to reconsideration.

Previous literature on clustering suggests that the clustering/two-phase FM methodology succeeds because of increased average node degree in the condensed netlist representation (cf., discussions in [5] and [23]). However, we have observed that good clusterings tend to cluster the densest parts of the hypergraph, leaving intact any nodes which are incident to "widely separated" regions of the hypergraph. The resulting clustered hypergraph can be less dense than the original hypergraph, suggesting that the clustering win is not so much related to density as it is to the reduction in problem size which allows most of the optimization effort to address the "difficult part" of the problem. Another assumption in previous work is that clusters should be of uniform size (e.g., [27, p. 243] or the original method of [5] which employs iterative matching). However, our work shows clearly that uniform cluster size need not be a dominant concern: Table IV shows some example profiles from a typical CAMS execution on the Primary2 benchmark ($|V| = 3014$). Even when there are 179 clusters, there exists a single cluster that contains over one-sixth of the modules. Thus, it seems acceptable for a clustering to group many tightly coupled nodes into a single large cluster, while other nodes remain singletons. The table also shows the total number of nets cut by the clustering, the

TABLE IV
DATA SHOWING TYPICAL CLUSTER DATA FOR A CAMS RUN ON BENCHMARK PRIMARY2 ASSUMING UNIT-AREA MODULES

Clustering Passes	Number of Clusters	Total Cuts	Total Pins	Scaled Cost	3 Largest Clusters		
					Size 1	Size 2	Size 3
Pass 1	603	1253	4937	0.000974	120	62	4
Pass 2	323	906	3501	0.000903	271	154	5
Pass 3	252	927	3240	0.000845	332	147	10
Pass 4	235	906	2967	0.000779	300	202	10
Pass 5	179	793	2320	0.000798	577	307	20

The table gives the total number of clusters, and the respective sizes of the largest, second largest and third largest clusters for each clustering. We also report total nets cut, total net-degree, and scaled cost values for each clustering.

sum of net degrees ("pins") over all clusters, and the scaled cost value of the clustering (a multiway generalization of ratio cut, due to Chan *et al.* [7]).

Although we have concentrated on the improved two-way partitionings afforded by CAMS, it is also of interest to characterize the clusterings generated by **Construct_Clustered_Netlist**, independent of their use in CAMS. As seen from Table IV, the CAMS clusterings tend to consist of two relatively large clusters and several clusters containing only one or two modules. This is not expected given that we construct the clustering from "good" two-way partitions which are likely to assign many modules similarly. Thus, CAMS clusterings may not be well suited for general clustering applications. It is possible that more "natural" clusterings may be obtained by using a CAMS-like methodology with an iterative multiway clustering approach such as that of Sanchis [28], but this is beyond the scope of the current work.

We also note that, in practice, CAMS finds its "optimum" solution after only a few iterations, seemingly before any effect of "adaptation" would have a chance to set in. This further suggests that CAMS relies more on problem size reduction than on the "big valley effect" which motivated the original AMS approach. Yet another contrast is that stochasticity in the original AMS approach can cause the new starting point to be very far from the "good" local minima, while the CAMS clustering guarantees a starting point whose structure closely reflects that of its parent solutions.

Last, we again note the clear connection between "adaptive multistart" and the concept of genetic algorithms. As described above, work in genetic algorithms by Mühlenbein *et al.* [25] has touched on ideas related to CAMS in addressing TSP. (The algorithm in [25] constructs a new more compact solution from *two* parents by clustering common substrings in the parents, i.e., in some sense, both the "problem reduction" and the "adaptation" elements are present.) However, our CAMS approach is basically "nongenetic" in that it uses "multiple parents," and does not rely on any sort of chromosomal representation of solutions.[8]

In conclusion, we have developed a new *clustered adaptive multistart* (CAMS) partitioning methodology which combines clustering-based two-phase FM with the adaptive multistart approach. For both the bisection and the ratio cut objectives, CAMS achieves partitioning results that are better than or as good as all of the best previous results in the literature. The method is very simple to state and implement, and relies on the present standard iterative approach, namely, the FM algorithm. CAMS is also highly stable in its rapid convergence to a good solution.

The success of CAMS for netlist bisection may make recursive bipartitioning-based placement a more appealing methodology. Moreover, the substantial improvement in solution quality may well put into question previous work [17] which compared "partitioning-based Rent parameters" of various algorithms, and found ratio-cut hierarchies to be superior. Finally, we note that CAMS is also naturally suited for clustering applications; our current work further applies the CAMS paradigm to VLSI placement, multiway netlist partitioning, and a number of other combinatorial formulations.

[8] Standard genetic algorithms seek a chromosomal representation which allows easy mutations between solution pairs; CAMS instead seeks more natural problem representations, and implicitly invokes very sophisticated recombination operators (e.g., for clustering and local search).

ACKNOWLEDGMENT

M. L. Smith provided the FM code used for the bisection experiments and the modified FM code used for the ratio cut experiments. Prof. C. K. Cheng and Dr. N.-C. Chou of the University of California at San Diego provided us with copies of their benchmark examples [9]. The anonymous reviewers provided many constructive suggestions and comments on the original draft of this paper. The work of A. B. Kahng was performed in part during a Spring 1993 sabbatical at the University of California at Berkeley; the hospitality of Prof. E. S. Kuh and his research group is gratefully acknowledged.

REFERENCES

[1] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing*. Norwell, MA: Kluwer, 1987.
[2] C. J. Alpert and A. B. Kahng, "Geometric embeddings for faster and better multi-way netlist partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, Dallas, TX, June 1993, pp. 743–748.
[3] K. D. Boese, A. B. Kahng, and S. Muddu, "New adaptive multistart techniques for combinatorial global optimizations," *Oper. Res. Lett.*, vol. 16, no. 2, pp. 101–113, 1994.
[4] R. M. Brady, "Optimization strategies gleaned from biological evolution," *Nature*, vol. 317, pp. 804–806, 1985.
[5] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser, "Graph bisection algorithms with good average case behavior," *Combinatorica*, vol. 7, no. 2, pp. 171–191, 1987.
[6] 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.*, 1989, pp. 775–778.
[7] P. K. Chan, M. D. F. Schlag, and J. Zien, "Spectral K-way ratio cut partitioning and clustering," in *Proc. Symp. Integrated Syst.*, Seattle, WA, Mar. 1993.
[8] C. K. Cheng and Y. C. Wei, "An improved two-way partitioning algorithm with stable performance," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 1502–1511, Dec. 1991.
[9] N.-C. Chou, private communication, Aug. 1993.
[10] J. Cong, L. Hagen, and A. B. Kahng, "Net partitions yield better module partitions," in *Proc. ACM/IEEE Design Automation Conf.*, Anaheim, CA, June 1992, pp. 47–52.
[11] J. Cong and M'L. Smith, "A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design," in *Proc. ACM/IEEE Design Automation Conf.*, Dallas, TX, June 1993, pp. 755–760.
[12] C.-L. Ding, C.-Y. Ho, and M. J. Irwin, "A new optimization driven clustering algorithm for large circuits" (extended abstract), in *Proc. 4th ACM/SIGDA Physical Design Workshop*, 1993, pp. 13–19.

[13] C. M Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175–181.
[14] L. Hagen and A. B. Kahng, "Fast spectral methods for ratio cut partitioning and clustering," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1991, pp. 10–13.
[15] ——, "New spectral methods for ratio cut partitioning and clustering," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 1074–1085, Sept. 1992.
[16] ——, "A new approach to effective circuit clustering," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1992, pp. 422–427.
[17] L. Hagen, A. B. Kahng, F. Kurdahi, and C. Ramachandran, "On the intrinsic rent parameter and new spectra-based methods for wireability estimation," in *Proc. European Design Automation Conf.*, Hamburg, Germany, Sept. 1992, pp. 202–208.
[18] L. W. Hagen and A. B. Kahng, "Method of solving discrete global optimization problems," U.S. Patent Appl. 08/296 808, patent pending (filing date: Aug. 1993).
[19] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation, Part I, Graph partitioning," *Oper. Res.*, vol. 37, pp. 865–892, 1989.
[20] D. Karger, "Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm," in *Proc. ACM–SIAM Symp. Discrete Algorithms*, Jan. 1993.
[21] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning of electrical circuits," *Bell Syst. Tech. J.*, Feb. 1970.
[22] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Comput.*, vol. C-33, pp. 438–446, May 1984.
[23] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. New York: Wiley-Teubner, 1990.
[24] H. Mühlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization," in *Proc. Int. Conf. Genetic Algorithms*, 1989, pp. 416–421.
[25] H. Mühlenbein, M. Georges-Schleuter, and O. Krämer, "Evolution algorithms in combinatorial optimization," *Parallel Comput.*, vol. 7, pp. 65–85, 1988.
[26] T.-K. Ng, J. Oldfield, and V. Pitchumani, "Improvements of a mincut partition algorithm," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1987, pp. 470–473.
[27] K. Roy and C. Sechen, "A timing-driven N-way chip and multi-chip partitioner," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1993, pp. 240–247.
[28] L. A. Sanchis, "Multiple-way network partitioning," *IEEE Trans. Comput.*, vol. 38, pp. 182–196, 1989.
[29] N. L. J. Ulder, E. H. L. Aarts, H.-J. Bandelt, P. J. M. van Laarhoven, and E. Pesch, "Genetic local search algorithms for the traveling salesman problem," in *Parallel Problem Solving from Nature*, H.-P. Schwefel and R. Männer, Eds. Berlin, Germany: Springer-Verlag, 1990, pp. 109–116.
[30] Y.C. Wei and C. K. Cheng, "Toward efficient hierarchical designs by ratio cut partitioning," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1989, pp. 298–301.
[31] ——, "A two-level two-way partitioning algorithm," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1990, pp. 516–519.

Lars W. Hagen (S'91–M'92) was born in Los Angeles, CA, in April 1963. He received the B.S. degree in engineering with option in computer science from California State University, Long Beach, and the M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles.

Since July 1994, he has been with Cadence Design Systems, Inc., San Jose, CA, where he is now a Member of Consulting Staff. His research interests include placement-based optimization for VLSI circuits with special emphasis on buffer tree generation for clock signals.

Andrew B. Kahng was born in San Diego, CA, in October 1963. He received the A.B. degree in applied mathematics/physics from Harvard University, Cambridge, MA, and the M.S. and Ph.D. degrees in computer science from the University of California at San Diego, La Jolla.

He joined the Computer Science Department at University of California, Los Angeles (UCLA), in 1989, and has been an Associate Professor since 1994. His interests include VLSI physical layout design and performance analysis, combinatorial and graph algorithms, and stochastic global optimization. Currently, he is Visiting Scientist (on leave from UCLA) at Cadence Design Systems, Inc., San Jose, CA.

Dr. Kahng received an NSF Research Initiation and Young Investigator Awards and a DAC Best Paper Award. He was the founding General Chair of the 1997 ACM/IEEE International Symposium on Physical Design, and defined the physical design roadmap as a member of the Design Tools and Test Working Group for the 1997 SIA National Technology Roadmap for Semiconductors.