A HYBRID MULTILEVEL/GENETIC APPROACH FOR CIRCUIT PARTITIONING

Charles J. Alpert, Lars W. Hagen† *and Andrew B. Kahng*

UCLA Computer Science Dept., Los Angeles, CA 90024-1596 USA

† Cadence Design Systems, San Jose, CA 95134 USA

ABSTRACT

We present a multilevel/genetic circuit partitioning algorithm that utilizes the Metis graph partitioning package [13], which had been previously applied to finite-element graphs. Our new technique produces better results than Metis alone, and also produces bipartitionings that are competitive with the recent methods of [17], [16] and [6] while using less CPU time.[1]

1. INTRODUCTION

A netlist hypergraph $H(V, E)$ has n modules $V = \{v_1, v_2, \ldots v_n\}$; a hyperedge (or *net*) $e \in E$ is defined to be a subset of V with size greater than one. A *bipartitioning* $P = \{X, Y\}$ is a pair of disjoint *clusters* (i.e., subsets of V) X and Y such that $X \cup Y = V$. The *cut* of a bipartitioning $P = \{X, Y\}$ is the number of nets which contain modules in both X and Y, i.e., $cut(P) = |\{e \mid e \cap X \neq \emptyset, e \cap Y \neq \emptyset\}|$. Given a balance tolerance r, the *min-cut bipartitioning problem* seeks a solution $P = \{X, Y\}$ that minimizes $cut(P)$ such that $\frac{n(1-r)}{2} \leq |X|, |Y| \leq \frac{n(1+r)}{2}$.

The standard bipartitioning approach is iterative improvement based on the Kernighan-Lin (KL) [14] algorithm, which was later improved by Fiduccia and Mattheyses (FM) [8]. The FM algorithm begins with some initial solution $\{X, Y\}$ and proceeds in a series of *passes*. During a pass, single modules are successively moved between X and Y until each module has been moved exactly once. Given a current solution $\{X', Y'\}$, the module $v \in X'$ (or Y') with highest *gain* ($= cut(\{X' - v, Y' + v\}) - cut(\{X, Y\})$) that has not yet been moved is moved from X' to Y'. After each pass, the best solution $\{X', Y'\}$ observed during the pass becomes the initial solution for a new pass, and the algorithm terminates when a pass does not improve the initial solution. FM has been widely adopted due to its short runtimes and ease of implementation.

A significant advance in KL-FM based methods was made by Krishnamurthy [15], who proposed a lookahead tie-breaking mechanism. Hagen et al. [9] recently showed that a "last-in-first-out" scheme based on the order that modules are moved in FM is significantly better than random or "first-in-first-out" tie-breaking schemes. Dutt and Deng [7] independently reached similar conclusions.

A second significant improvement to FM integrates *clustering* into a "two-phase" methodology. A k-way clustering of $H(V, E)$ is a set of disjoint clusters $P^k = \{C_1, C_2, \ldots C_k\}$ such that $C_1 \cup C_2 \cup \ldots \cup C_k = V$ where k is sufficiently large.[2] We denote the input netlist as $H_0(V_0, E_0)$. A clustering $P^k = \{C_1, C_2, \ldots, C_k\}$ of H_0 induces the *coarser* netlist $H_1(V_1, E_1)$, where $V_1 = \{C_1, C_2, \ldots, C_k\}$ and for every $e \in E_0$, the net e' is a member of E_1 where $e' = \{C_i \mid \exists v \in e \text{ and } v \in C_i\}$ unless $|e'| = 1$ (i.e., each cluster in e' contains some module that is in e). In two-phase FM, a clustering of H_0 induces the coarser netlist H_1, and then FM is run on $H_1(V_1, E_1)$ to yield the bipartitioning $P_1 = \{X_1, Y_1\}$. This solution then *projects* to the bipartitioning $P_0 = \{X_0, Y_0\}$ of H_0, where $v \in X_0(Y_0)$ if and only if for some $C_h \in V_1$, $v \in C_h$ and $C_h \in X_1(Y_1)$. FM is then run a second time on $H_0(V_0, E_0)$ using P_0 as the initial solution.

Many clustering algorithms for two-phase FM have appeared in the literature (see [2] for an overview of clustering methods and for a general netlist partitioning survey), e.g., Bui et al. [4] find a random maximal matching in the netlist and compact the matched pairs of modules into $\frac{n}{2}$ clusters, which can then be repeated. Often, two-phase FM (not including the time needed to cluster) is faster than a single FM run because the first FM run is for a smaller netlist and the second FM run starts with a good initial solution, which allows fast convergence.

The "two-phase" approach can be extended to include more phases; such a *multilevel* approach is illustrated in Figure 1. In a multilevel algorithm, a clustering of the initial netlist H_0 induces the coarser netlist H_1, then a clustering of H_1 induces H_2, etc. until the coarsest netlist H_m is constructed ($m = 4$ in the Figure). A partitioning solution $P_m = \{X_m, Y_m\}$ is found for H_m (e.g., via FM) and this solution is projected to $P_{m-1} = \{X_{m-1}, Y_{m-1}\}$. P_{m-1} is then refined, e.g., by using it as an initial solution for FM. In the Figure, each projected solution is indicated by a dotted line and each refined solution is given by a solid dividing line. This *uncoarsening* process continues until a partitioning of the original netlist H_0 is derived.

Multilevel clustering methods have not been thoroughly explored in the physical design literature, with the exception of [11]. Cong and Smith [5] applied multilevel techniques to partition clique-based clusters and Sun and Sechen [18] have used (three-level) multilevel clustering in their cell placement algorithm. However, multilevel partitioning is very well-studied in the scientific computing community, with two strong public domain software packages having been developed. Hendrickson and Leland [12] developed the Chaco partitioning package which utilizes both spectral and iterative techniques. Karypis and Kumar [13] later developed a similar package called Metis, which also allows non-recursive multi-way partitioning. Our work can be viewed as a "wrapper" around the Metis package, with the use of Metis as opposed to Chaco being completely arbitrary.

The Metis package [13] has produced good partitioning results for finite-element graphs, and is extremely efficient. Our initial hypothesis, which our work has verified, was that Metis adapted to circuit netlists is both better and faster than FM. We have also integrated Metis into a genetic algorithm; our experiments show that this ap-

---

[1] CJA is now with IBM Austin Research Laboratory, Austin TX, e-mail: alpert@austin.ibm.com. LWH e-mail: lars@cadence.com. ABK is currently Visiting Scientist (on sabbatical from UCLA) at Cadence Design Systems, San Jose CA, e-mail: ak@cadence.com.

[2] A *partitioning* and a *clustering* are identical by definition, but the term partitioning is generally used when k is small (e.g., $k \leq 10$), and the term clustering is generally used when k is large (e.g., $k = \Theta(n)$ with constant average cluster size). Although a bipartitioning can also be written as $P^2 = \{C_1, C_2\}$, we use the notation $P = \{X, Y\}$ to better distinguish between partitioning and clustering.

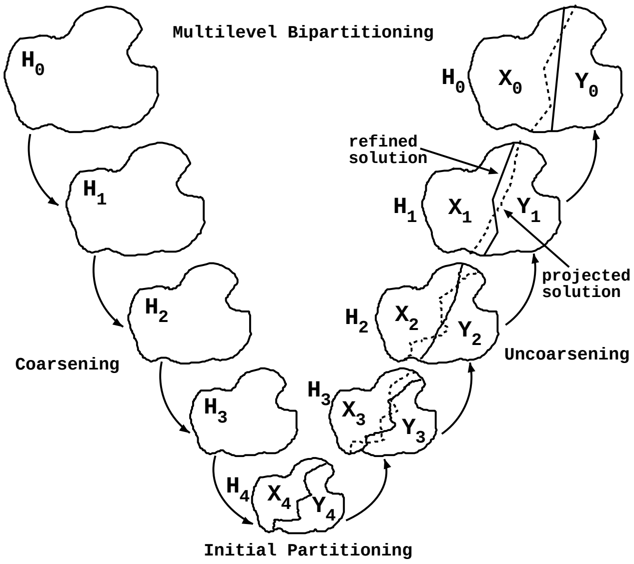

Figure 1. The multilevel bipartitioning paradigm.

proach produces better average and minimum cuts than Metis alone. Overall, we generate bipartitionings that are competitive with the recent approaches of [17] [16] [6] while requiring much less CPU time.

2. GRAPH PARTITIONING USING METIS

The Metis package [13] has multiple algorithm options for coarsening, for the initial partitioning step, and for refinement, e.g., one can choose among eight distinct matching-based clustering schemes. Our methodology follows the recommendations of [13]. Before multilevel partitioning is performed, the adjacency lists for each module are randomly permuted. The following discussion applies our previous notation to weighted graphs; a weighted graph is simply a hypergraph H_i with $|e| = 2$ for each $e \in E_i$ and a nonnegative weight function w on the edges.

To cluster, Karypis and Kumar suggest *Heavy-Edge Matching* (HEM), a variant of random matching [4]. A matching M of H_i is a subset of E_i such that no module is incident to more than one edge in M. Each edge in the matching is contracted to form a cluster; contracted edges should have highest possible weight since they will not be cut in the graph H_{i+1}. HEM visits the modules in random order; if a module u is unmatched when visited, the edge (u, v) is added to M where $w(u, v)$ is maximum over all unmatched modules v; if u has no unmatched neighbors, it remains unmatched. This greedy algorithm is suboptimal, but runs in $O(|E_i|)$ time.

An initial bipartitioning for H_m is formed by the Greedy Graph Growing Partitioning (GGGP) algorithm. Initially, one "fixed" module v is in its own cluster X_m and the rest of the modules are in Y_m. Modules with highest gains are greedily moved from Y_m to X_m until $P_m = \{X_m, Y_m\}$ satisfies the cluster size constraints. Despite its simplicity, the GGGP heuristic is as effective as other heuristics when partitioning finite-element graphs [13].

Refinement uses the *Boundary Kernighan-Lin Greedy Refinement* (BGKLR) scheme. The heuristic uses the FM single-module neighborhood structure. Kumar and Karypis label the KL algorithm "greedy" when only a single pass is performed, and propose a hybrid algorithm which performs "complete" KL when the graph is small (i.e., less than 2000 modules) and greedy KL for larger graphs, thereby saving substantial CPU time. A time-saving "boundary" scheme (as in [12]) is used to update gains: only modules incident to cut edges (i.e., boundary modules) are stored in the FM bucket data structure and are eligible to be moved. The overall Metis methodology is shown in Figure 2.

The Metis Algorithm
Input: Graph $H_0(V_0, E_0)$
Output: Bipartitioning $P_0 = \{X_0, Y_0\}$

1. i = 0; randomly permute the adjacency lists of H_0.
2. **while** $|V_i| < 100$ **do**
3. Use HEM to find a matching M of H_i
4. Contract each edge in M to form a clustering. Construct the coarser graph $H_{i+1}(V_{i+1}, E_{i+1})$. Set i to $i + 1$.
5. Let $m = i$. Apply GGGP to H_m to derive P_m. Refine P_m using BGKLR.
6. **for** i = m-1 **downto** 0 **do**
7. Project solution P_{i+1} to the new solution P_i. Refine P_i using BGKLR.
8. **return** P_0.

Figure 2. The Metis Algorithm

To run Metis on circuit netlists, we must first construct a sparse graph from a given circuit hypergraph. The traditional clique net model (which adds an edge to the graph for every pair of modules in a given net) is a poor choice since large nets destroy sparsity. Since we observed that keeping large nets generally increases the cut size, our method removes all nets with more than 50 modules. For each net e, our converter picks $5 \cdot |e|$ random pairs of modules in e and adds an edge with cost one into the graph for each pair. Our converter retains the sparsity of the circuit, introduces randomness and is fairly efficient.

3. A GENETIC VERSION OF METIS

Our experiments show that over multiple independent runs, Metis will generate at least one very good solution but has unstable performance. To remedy this, we have integrated Metis into a genetic framework (see [10]).

An *indicator vector* $\vec{p} = \{p_1, p_2, \ldots, p_n\}$ for a bipartitioning $P = \{X, Y\}$ has entry $p_i = 0$ if $v_i \in X$ and entry $p_i = 1$ if $v_i \in Y$, for all $i = 1, 2, \ldots, n$. The *distance* between two bipartitionings P and Q with corresponding indicator vectors $\vec{p}$ and $\vec{q}$ is given by $\sum_{i=1}^{n} |p_i - q_i|$, i.e., by the number of module moves needed to derive solution Q from the initial solution P. Boese et al. [3] showed that the set of local minima generated by multiple FM runs exhibit a "big valley" structure: solutions with smallest distance to the lowest-cost local minima also have low cost, and the best local minima are "central" with respect to the other local minima. Thus, we seek to combine several local minimum solutions generated by Metis into a more "central" solution.

Given a set S of s solutions, the s-digit binary code $C(i)$ for module v_i is generated by concatenating the i^{th} entries of the indicator vectors for the s solutions. We construct a clustering by assigning modules v_i and v_j to the same cluster if $C(i)$ and $C(j)$ are the same code. Our strategy integrates this code-generated clustering into Metis, in that we use HEM clustering and force every clustering generated during coarsening to be a refinement of the code-based clustering.[3] Our Genetic Metis (GMetis) algorithm is shown in Figure 3.

[3] A clustering P^k is a *refinement* of Q^l ($k \geq l$) if some division of clusters in Q^l will yield P^k.

The Genetic Metis (GMetis) Algorithm

Input: Hypergraph $H(V,E)$ with n modules
Output: Bipartitioning $P = \{X,Y\}$

Variables: s: Number of solutions
$numgen$: Number of generations
$C(i)$: s-digit code for module v_i
S: set of the s best solutions seen
G: graph with n modules

1. Set $C(i) = 00 \ldots 0$ for $1 \leq i \leq n$
2. **for** i = 0 **to** $numgen - 1$ **do**
3.   **for** j = 0 **to** $s - 1$ **do**
4.   **if** $((i \cdot s) + j)$ modulo $10 = 0$ **then** convert H to graph G
5.   P = Metis(G) (HEM based on codes $C(i)$)
6.   **if** $\exists Q \in S$ such that Q has larger cut than P **then** $S = S + P - Q$.
7.   **if** $i > 0$ **and** $(s(i-1) + j)$ modulo $5 = 0$ **then** recompute $C(i)$ for $1 \leq i \leq n$ using S.
8. **return** $P \in S$ with lowest cut.

Figure 3. The Genetic Metis Algorithm

Step 1 initially sets all codes to $00 \ldots 0$, which causes GMetis to behave just like Metis until s solutions are generated. Steps 2 and 3 are loops which cause $numgen$ generations of s solutions to be computed. Step 4 converts the circuit hypergraph into a graph; this step is performed only once out of every 10 times Metis is called. We perform the conversion with this frequency to reduce runtimes while allowing different graph representations. In Step 5, Metis is called using our version of HEM described above. Step 6 maintains the set of solutions S; we replace solution $Q \in S$ with solution P if the cut of P is smaller than that of Q. Step 7 computes the binary code for each module based on the current solution set, but only after the first generation has completed and five solutions with the previous code-based clustering have been generated. The solution with lowest cut is returned in Step 8.

4. EXPERIMENTAL RESULTS

All of our experiments use a subset of the benchmarks from the ACM/SIGDA suite; hypergraph formats and statistical information for these circuits are available on the world wide web at http://ballade.cs.ucla.edu/~cheese. Our experiments assume unit module areas, and our code was written in C++ and was compiled with g++ $v2.4$ on a Unix platform. Our experiments were run on an 85 MHz Sun Sparc 5 and all runtimes reported are for this machine (in seconds) unless otherwise specified.

Our first set of experiments compares Metis against both FM and two-phase FM. We ran Metis 100 times with balance parameter $r = 0$ (exact bisection) and recorded the minimum cut observed in the second column of Table 1. Since there are many implementations of FM (some of which are better than others), we compare to the best FM results found in the literature.

Dutt and Deng [6] have implemented very efficient FM code; their exact bisection results for the best of 100 FM runs are given in the third column of Table 1 and the corresponding Sparc 5 run times are given in the last column. The fourth column reports the FM results of [9] which use a LIFO tie-breaking strategy and a lookahead function that improves upon that of [15]. Finally, the fifth column gives the best two-phase FM results observed for various clustering algorithms as reported in [1] and [9].

Metis does not appear to be faster than FM for circuits with less than 2000 modules, but for larger circuits with 5000-12000 modules, Metis is 2-3 times faster. With regard to solution quality, we conclude that multilevel approaches are unnecessary for small circuits, but greatly enhance solution quality for larger circuits. For large circuits, more than two levels of clustering are needed if the iterative approach is to be effective.

Test Case	Minimum cut (100 runs) Metis	FM [6]	FM [9]	2-FM [1] [9]	CPU (s) Metis	FM [6]
balu	34	32			23	21
bm1	53	55		52	22	24
primary1	55	57	56	53	22	24
test04	53	86		56	37	41
test03	61	72		60	39	56
test02	99	115		97	43	46
test06	94	71		68	53	50
struct	36	45	36	43	41	46
test05	107	97		93	69	81
19ks	116	142		121	59	115
primary2	158	236	171	182	90	128
s9234	49	53			72	222
biomed	83	83	83	124	134	296
s13207	84	92			111	339
s15850	62	112			123	339
industry2	218	428	275	438	349	727
industry3	292		312	328	399	
avqsmall	175		373	399	293	
avqlarge	171		406	518	355	

Table 1. Comparison of Metis with FM.

Test Case	Metis min	Metis avg	Metis CPU	GMetis min	GMetis avg	GMetis CPU
balu	34	47	26	32	38	24
bm1	53	65	23	54	59	22
primary1	55	66	23	55	59	21
test04	53	68	37	52	58	37
test03	61	76	42	65	74	39
test02	99	113	44	96	101	42
test06	94	117	59	97	121	55
struct	36	52	41	34	40	39
test05	107	125	70	109	117	69
19ks	116	132	59	112	116	59
primary2	158	195	95	165	174	91
s9234	49	66	71	45	52	68
biomed	83	149	145	83	134	143
s13207	84	90	106	78	89	112
s15850	62	84	126	59	74	125
industry2	218	280	336	204	230	339
industry3	292	408	384	291	313	423
s35932	55	71	257	56	62	265
s38584	55	101	310	53	67	368
avqsmall	175	241	289	148	174	322
s38417	73	110	294	74	104	301
avqlarge	171	248	318	144	181	355

Table 2. Comparison of Metis with Genetic Metis.

The next set of experiments compares Metis with GMetis. We ran GMetis for 10 generations while maintaining $s = 10$ solutions so that both Metis and GMetis considered 100 total solutions. The minimum and average cuts observed, as well as total CPU time, are reported for both algorithms in Table 2. On average, GMetis yields minimum cuts that are 2.7% lower than Metis, and significantly lower average cuts. We believe that GMetis can have its greatest impact for larger circuits.

Finally, we compare GMetis to other recent partitioning works in the literature, namely PROP [6], Paraboli [17], and GFM [16], the results of which are quoted from the original sources and presented in Table 3. All these works use $r = 0.1$, i.e., each cluster contains between 45%

Test Case	Cuts			
	PROP	Paraboli	GFM	GMetis(bal)
balu	27	41	27	27(32)
bm1	50			48(53)
primary1	47	53	47	47(54)
test04	52			49(52)
test03	59			62(66)
test02	90			95(96)
test06	76			94(93)
struct	33	40	41	33(34)
test05	79			104(109)
19ks	105			106(110)
primary2	143	146	139	142(158)
s9234	41	74	41	43(45)
biomed	83	135	84	102(83)
s13207	75	91	66	74(70)
s15850	65	91	63	53(60)
industry2	220	193	211	177(204)
industry3		267	241	243(286)
s35932		62	41	57(55)
s38584		55	47	53(53)
avqsmall		224		144(145)
s38417		49	81	69(77)
avqlarge		139		145(144)

Table 3. Cut comparisons of GMetis with PROP, Paraboli, and GFM allowing 10% deviation from bisection. Exact bisection results for GMetis are given in parentheses.

Test Case	CPU			
	PROP	Paraboli	GFM	GMetis
balu	16	16	24	14
bm1	20			12
primary1	19	18	16	12
test04	49			21
test03	51			23
test02	64			26
test06	75			32
struct	42	35	80	27
test05	97			46
19ks	87			39
primary2	139	137	224	53
s9234	139	490	672	58
biomed	250	711	1440	95
s13207	177	2060	1920	102
s15850	291	1731	2560	114
industry2	867	1367	4320	245
industry3		761	4000	299
s35932		2627	10160	266
s38584		6518	9680	397
avqsmall		4099		328
s38417		2042	11280	281
avqlarge		4135		417

Table 4. CPU Time Comparisons.

and 55% of the total number of modules. Table 4 quotes the CPU times in seconds for PROP, Paraboli, and GFM on a Sun Sparc 5, a DEC 3000 Model 500 AXP, and a Sun Sparc 10 respectively. We modified GMetis to handle varying size constraints but found that GMetis with $r = 0.1$ was sometimes outperformed by GMetis with $r = 0$ (exact bisection). Hence, in Table 3, we present results for GMetis with $r = 0.1$ and $r = 0$ (given in parentheses). We report runtimes for GMetis for $r = 0$. These experiments used $s = \log_2 n$ ($|V| = n$) solutions and 12 generations. Observe that GMetis cuts are competitive with the other methods, especially for several larger benchmarks. The big win for GMetis is its short runtime, e.g., generating a single solution for avqlarge takes $417/(12 \log_2 25178) = 2.5$ seconds.

In conclusion, we have integrated the Metis multilevel partitioning package of [13] into a genetic algorithm. We show that (i) Metis outperforms previous FM-based approaches, (ii) GMetis improves upon Metis for large benchmarks, and (iii) GMetis is competitive with previous methods while using less CPU. We have recently implemented our own hypergraph multilevel algorithm and integrated it into a new cell placement tool.

REFERENCES

[1] C. J. Alpert and A. B. Kahng, "A General Framework for Vertex Orderings, with Applications to Netlist Clustering", *IEEE Trans. on VLSI* 4(2), 1996, pp. 240-246.

[2] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey", *Integration, the VLSI Journal*, 19(1-2), 1995, pp. 1-81.

[3] K. D. Boese, A. B. Kahng, and S. Muddu, "A New Adaptive Multi-Start Technique for Combinatorial Global Optimizations", *Operations Research Letters*, 16(20), 1994, pp. 101-13.

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

[5] J. Cong and M. Smith, "A Parallel Bottom-up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Design" *Proc. ACM/IEEE DAC*, 1993, pp. 755-760.

[6] S. Dutt and W. Deng, "A Probability-Based Approach to VLSI Circuit Partitioning", *Proc. ACM/IEEE DAC*, 1996, pp. 100-105.

[7] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques", *Technical Report*, Department of Electrical Engineering, University of Minnesota, Nov. 1995.

[8] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions", *Proc. ACM/IEEE DAC*, 1982, pp. 175-181.

[9] L. W. Hagen, D. J.-H. Huang, and A. B. Kahng, "On Implementation Choices for Iterative Improvement Partitioning Algorithms", *Proc. European Design Automation Conf.*, 1995, pp. 144-149.

[10] L. W. Hagen and A. B. Kahng, "Combining Problem Reduction and Adaptive Multi-Start: A New Technique for Superior Iterative Partitioning", to appear in *IEEE Trans. on CAD.*

[11] S. Hauck and G. Borriello, "An Evaluation of Bipartitioning Techniques", *Proc. Chapel Hill Conf. on Adv. Research in VLSI*, 1995.

[12] B. Hendrickson and R. Leland, "A Multilevel Algorithm for Partitioning Graphs", Technical Report SAND93-1301, Sandia National Laboratories, 1993.

[13] G. Karypis and V. Kumar, "Unstructured Graph Partitioning and Sparse Matrix Ordering", *Technical Report*, CS Dept., Univ. of Minnesota, 1995.

[14] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell Systems Tech. J.*, 49(2), 1970, pp. 291-307.

[15] B. Krishnamurthy, "An Improved Min-Cut Algorithm for Partitioning VLSI Networks", *IEEE Trans. Computers*, 33(5), 1984, pp. 438-446.

[16] L.-T. Liu, M.-T. Kuo, S.-C. Huang, and C.-K. Cheng, "A Gradient Method on the Initial Partition of Fiduccia-Mattheyses Algorithm", *Proc. IEEE Intl. Conf. Computer-Aided Design*, 1995, pp. 229-234.

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

[18] W. Sun and C. Sechen, "Efficient and Effective Placements for Very Large Circuits" *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1993, pp. 170-177.