Fast Hypergraph Partition

Andrew B. Kahng
Department of Computer Science
University of California, San Diego
La Jolla, CA 92093

Abstract

We present a new $O(n^2)$ heuristic for hypergraph min-cut bipartitioning, an important problem in circuit placement. Fastest previous methods for this problem are $O(n^2 \log n)$. Our approach is based on the *intersection graph* G dual to the input hypergraph. Paths in G are used to construct *partial* bipartitions which can be completed optimally. The method is provably good and, in particular, obtains optimum results for "difficult" inputs, i.e., hypergraphs with smaller than expected minimum cutsize. Computational results for a wide range of inputs are also discussed.

1. Introduction

In VLSI/PCB CAD, a circuit netlist naturally defines a hypergraph H, with vertices corresponding to modules and edges corresponding to signal nets. Some useful definitions concerning hypergraphs are:

- A hypergraph H has vertex set $V = \{v_1, v_2, \ldots, v_m\}$ and edge set $E = \{e_1, e_2, \ldots, e_n\}$, with each edge a subset of V, i.e., $e_j \subseteq V$, $j = 1,...,n$. If $|e_j| = 2 \ \forall j$, H is also a graph.
- A *cut* in H is a partition of V into two disjoint nonempty sets V_L and V_R. An edge *crosses* the cut if it contains vertices $v_1 \in V_L$ and $v_2 \in V_R$. The *size* of a cut is the number of edges that cross the cut.
- A *bisection* of H is a cut which satisfies $|V_L| - |V_R| \leq 1$. The minimum bisection, or *min-cut bisection*, of H is the bisection with minimum size.

A large body of work confirms hypergraph min-cut bisection as a good objective for VLSI and PCB clustering placement in applications ranging from standard-cell IC design to two-sided board technologies [23]. Breuer [4] proposed three min-cut placement heuristics using the netlist hypergraph and a bounding-box net model. Following Schweikert and Kernighan [22], Breuer's methods rely on Kernighan-Lin [17] partitioning of the circuit hypergraph. Improvements to the min-cut scheme include the "cell gain" metric of Fiduccia and Mattheyses [9], the terminal propagation method of Dunlop and Kernighan [8], and the work of Carter et al. [6]. More recently, graph space mappings [11], network flow [7] and linear programming [16] methods have been proposed. The hypergraph min-cut bisection problem is NP-complete [12].

Kernighan-Lin variants are attractive for their ease of implementation and $O(n^2 \log n)$ complexity, but this bound holds only for the lowest-level (*2-opt*) implementation. Finding *k-opt* cuts with $k \geq 3$ is more expensive. Stochastic (e.g. annealing), network flow, graph space and linear programming methods generally yield good results for hypergraphs derived from logic circuits, but their $O(n^3)$ or higher complexity render them impractical for large problem instances [14][19].

In practice, there is little reason to insist that the numbers of nodes on either side of the cut be exactly equal. Since nodes of a hypergraph can represent physical modules with widely varying attributes (e.g., area, shape, technology), relaxation of the bisection criterion is acceptable. [9] proposed an r-bipartition metric, where the difference in node cardinalities is maintained at r or less. Fukunaga et al. and others [11][14] have introduced penalty terms in the placement metric to reflect imbalanced V_L and V_R. Such cost functions are very natural, as the true goal of circuit partitioning is preservation of logical clustering, rather than actual bisection of the netlist. The culmination of this trend is the recent introduction in [20] of a *quotient* cut objective: $\min\limits_{C} \frac{e(V_L, V_R)}{|V_L| \cdot |V_R|}$, with C denoting all possible cuts and e denoting the size of a cut.

Hartoog [15] has noted that no one algorithm in the literature consistently gives good results; even annealing has a large variance in performance. A crucial observation concerns the fact that all of the above approaches will perform *arbitrarily* poorly on hypergraphs with smaller than expected minimum cutsize, i.e., they have no error bound. Since heuristics in the CAD literature generally give acceptable performance, this seems to imply that circuit partitioning problems are in general "easy". However, in an "easy" problem instance, even a random cut will differ from the optimum cut by at most a constant factor [2], and thus we feel that it is important to find a heuristic which performs significantly better than random. Bui et al. [5] discuss the role of "difficult" (i.e., harder than random) inputs in distinguishing a good heuristic.

In this paper, we propose a new, provably good $O(n^2)$ method for hypergraph partitioning. Previous fastest methods are $O(n^2 \log n)$. Our method is based on the dual *intersection graph* of the input netlist. The heuristic uses random longest breadth-first search (BFS) paths (or diameters) within the intersection graph to construct partial hypergraph bipartitions in $O(n^2)$ time. Such a construction is expected to place all but a constant proportion of the nodes in H. Given this initial partial

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1989 ACM 0-89791-310-8/89/0006/0762 $1.50

bipartition, we can complete the partition *optimally* in $O(n \log n)$ time. In the section below, we develop the algorithm; the following section treats some theoretical aspects of our method.

Section 4 briefly compares performance of the method with popular heuristics described in [4], [8], [14] and [18]. We close by mentioning possible algorithm enhancements and areas for further research.

2. Finding a Min-Cut Bipartition in a Hypergraph

The Intersection Graph.

We begin by dualizing the problem. Given a hypergraph H, consider the graph $G = (V,E)$ which has n vertices, one for each edge of H (i.e., each signal in the netlist), with two vertices adjacent if and only if the corresponding edges in H intersect (i.e., the signals have some module in common). G is sometimes referred to as the ***intersection graph*** of H. Note that for a given H, G is well-defined; however, there is no unique reverse construction. If necessary, to avoid confusion we will sometimes call a vertex of H an "H-vertex", with "G-vertex" defined similarly. Figure 1 shows a hypergraph with 8 nodes and 5 hyperedges (labeled A - E) along with its associated intersection graph.

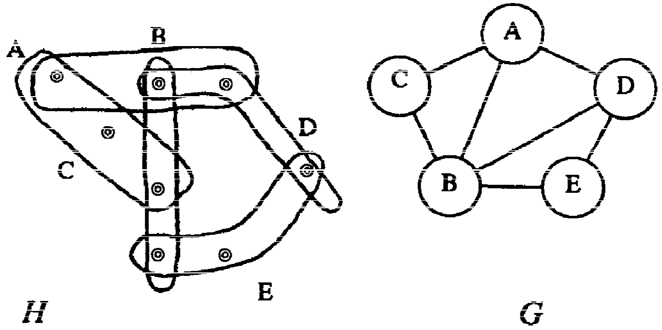

Figure 1

We now reformulate the min-cut partition of H in terms of G, as follows. Notice that an arbitrary cut through G will divide the vertices of G into disjoint sets V_L and V_R. Vertices adjacent to the cut, i.e., those elements of V_L adjacent to elements of V_R (and vice versa) are called *boundary* nodes. We denote the set of boundary nodes by B, with B_L and B_R defined in the obvious way. Every node of V_L (V_R) which is not in B represents an edge in the original hypergraph H all of whose vertices are placed in the left (right) half of the partition. In other words, by construction nodes which are not in B represent signals that do not cross the cut.

Definition: A *partial bipartition* of $G = (V,E)$, denoted by (G, V_l, V_r), consists of disjoint sets $V_l, V_r \subset V$ with no edge $e_{ij} \in E$ satisfying $v_i \in V_l$, $v_j \in V_r$.

Thus, any cut in G naturally induces a partial bipartition.

Definition: A *completion* of a partial bipartition (G, V_l, V_r) consists of disjoint sets $V_L, V_R \subset V$ with $V_l \subset V_L$, $V_r \subset V_R$ and $V_L \cup V_R = V$.

Figure 2 depicts the structure of such an intersection graph.

An arbitrary cut in G cannot in general be interpreted as a cut in H. However, the non-boundary nodes of a cut in G form a partial bipartition which can be *completed* such that the final cut in G corresponds to a valid cut in H. In other words, we use the graph cut in G to obtain a "handle" on the original hypergraph partition problem. Non-boundary vertices in G implicitly partition the vertices in H belonging to the corresponding hyperedges. To complete the partition, we need only partition the H-vertices which belong to boundary nodes in G, such that the final cutsize is minimum. Given the initial partial bipartition, it turns out that there is a simple method for assignment of boundary vertices to the two halves of the partition. The method is similar to heuristics for, e.g, PLA folding [13].

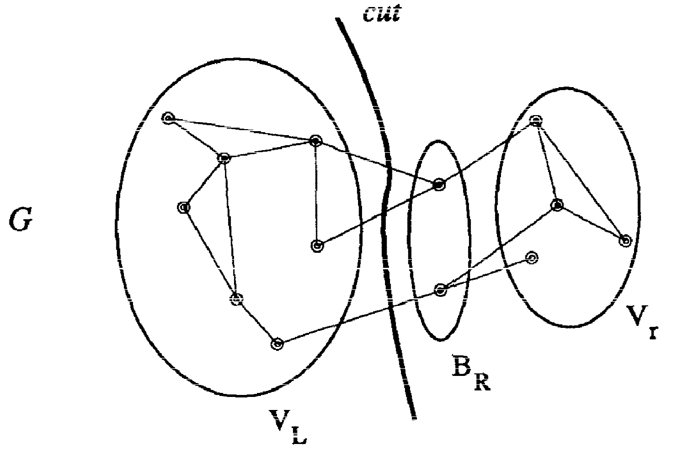

Figure 2

Partitioning the Boundary Set.

Recall that the initial graph cut in G implicitly partitions B into disjoint nonempty sets B_L and B_R. Consider the subgraph of G induced by B, with all e_{uv} edges deleted, where $u,v \in B_L$ or $u,v \in B_R$. This graph, which we call the boundary graph or G', is *bipartite*. In the optimal completion of the hypergraph partition, each node of G' (which is an edge in H) will either cross the cut or not cross the cut. Nodes that cross the cut are called *losers*, while nodes whose modules are all on one side of the cut are *winners*.

Fact: If $v \in B$ is a *winner*, then all nodes adjacent to v in the bipartite graph G' are *losers*.

It follows that minimizing the number of losers gives the hypergraph min-cut completion of the original graph cut. ($\frac{|B|}{2}$ is thus a trivial upper bound on the cutsize.) Our method for completing the partition is as follows:

Complete_Cut:

<1> If G' is not the trivial graph, select the vertex v in G' with minimum degree and mark it as a winner.

<2> Mark all nodes in G' adjacent to v as losers.

<3> Delete v, the losers and their incident edges from G'.

Figure 3 shows an example of the boundary graph; the winners are nodes $\{b,c,e,f,h,i\}$. Our next result can be proven by induction.

Theorem: Given a graph cut in G and an associated initial partial bipartition which induces a connected graph G', the *Complete_Cut* method yields a cutsize within one of the optimum completion of the bipartition. □

This means that once we find a cut with non-empty node sets on either side of the boundary, we can essentially finish the partition optimally. (In practice, a graph cut can be obtained by doing breadth-first search from two distant nodes of G until the two expanding sets meet to define a cutline.)

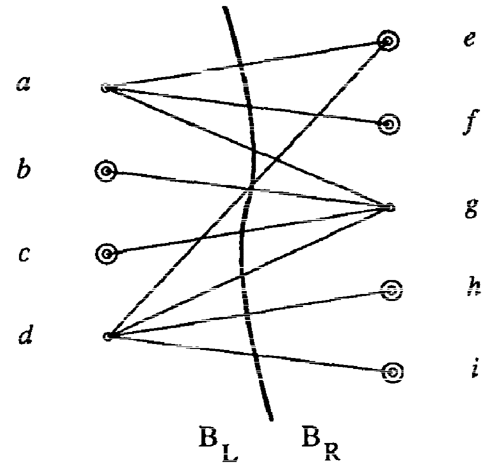

Figure 3

The Basic Algorithm.

Our algorithm for hypergraph bipartition can now be sketched.

Algorithm I:

Given an input hypergraph in the form of an intersection graph G:

<1> Pick an arbitrary node u in G and use BFS to find a node v furthest from u ($O(n^2)$ time)

<2> Generate an initial cut in G using BFS from u and v, and determine boundary set B ($O(n^2)$ time)

<3> Partition B between L and R sides, using the heuristic in Section 2.2 above ($O(n \log n)$ time)

Example.

Let N be the following netlist:

Signal	Modules
a	1,2,11
b	2,4,11
c	1,3,4,8
d	4,8
e	2,4,8
f	3,5,6,7
g	3,5,6,7
h	5,7,8
i	6,9,10
j	6,7,9,10
k	9,10
l	11,12

Thus, the nodes of the input hypergraph H are labeled $1, 2, \ldots, 12$, while the hyperedges are denoted by a, b, etc. The resulting intersection graph G is shown in Figure 4.

We find that nodes k and l form a "furthest-removed" pair in G. After breadth-first search, the boundary set is determined to be $\{c,d,e,f,g,h\}$, and the corresponding initial partial bipartition separates modules $\{1,2,4,11,12\}$ from modules $\{6,7,9,10\}$. After processing the bipartite graph G', we see that $\{d,e,f,g\}$ are winners. The final partition separates $\{1,2,4,8,11,12\}$ from $\{3,5,6,7,9,10\}$. Signals c and h are the only ones to cross the cut, so the crossing count is 2.

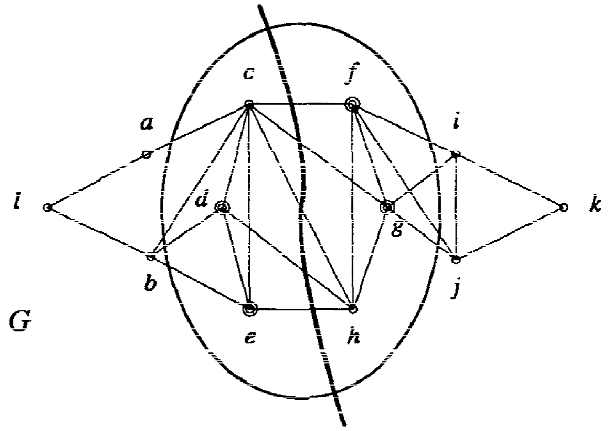

Figure 4

Algorithm I requires $O(n^2)$ processing time, where n is the number of edges in the original hypergraph. In circuit design, average net size is generally constant for a given design methodology and technology, so n scales with the number of modules in the netlist (order of the input hypergraph). The most interesting feature of Algorithm I is that despite its simplicity, it is provably good.

3. Discussion

Implementation Issues and the Graph Model.

Probabilistic analysis of the algorithm relies on bounded degree of nodes in G. In practice, some placement algorithms are ill-suited to the recent trend of higher average signal size (e.g., bus-driven layout). This is usually because of the particular net model (complete graph, k-star, MRST) chosen. Thus, many CAD programs will ignore signals larger than a heuristic threshold during placement. We ask: Are such relaxation schemes valid?

In practice, chips have a bounded number of pins, VLSI standard cells have a small, bounded number of ports, and so forth. A simple probabilistic argument shows the following:

Theorem: In a random hypergraph H, if an edge e has degree k, e will traverse the min-cut bipartition with probability $1 - O(2^{-k})$. □

This result was verified empirically for industry IC netlists by examining the effect of large signals on the heuristic partition. Simulated annealing (Table 1) and Kernighan-Lin results show that in the best heuristic partition, large signals ($k \geq 14$) almost always contribute to the cut value.

Our analysis shows that we can ignore signals above a size threshold as low as $k \geq 10$ with very small expected error in cutsize. Accordingly, we heuristically ignore large edges in the input hypergraph. This allows us to assume that G has bounded degree in the theoretical analysis below. Furthermore, in practice we find that the sparser hypergraph will have greater graph diameter of G, so the size of the boundary set is smaller.

Technology	$k \geq 20$ crossing %	$k \geq 14$ crossing %	$k \geq 8$ crossing %
PCB	99+	98	97+
Std-Cell	99+	97	96+
Hybrid	99+	99	97+

Table 1

Table 1. Results averaged over 10 simulated annealing runs for each example in the industry test suite.

Probabilistic Analysis.

To examine the case of bounded-degree G, we consider hypergraphs $H(n,d,r,c)$ with n nodes, node degree $\leq d$, edge degree $\leq r$, and minimum cutsize c. (This naturally fits such paradigms as circuit layout.) For large random hypergraphs with small r, as with random graphs, expected minimum cutsize is $\Theta(|E|)$, so even random cuts are expected to be suboptimal by only a constant factor. It is thus useful to evaluate performance of a bipartitioning heuristic on those "difficult" inputs which have smaller than expected minimum cutsize.

Following Bui et al. [5], we consider the class $H(n,d,r,c)$ with $c = o(n^{1-1/d})$, $n \rightarrow \infty$. Our main result follows from expander properties of H and the associated G.

Theorem: As $n \rightarrow \infty$, for almost all H in $H(n,d,r,c)$ with $c = o(n^{1-1/d})$, Algorithm I will output the min-cut bipartition of H. □

Essentially, the hyperedges in the minimum cut of H will almost always be nodes of the boundary graph G'; if the cut is unique, then precisely those nodes will have high degree in G' and become losers.

Finding Graph Diameters.

One may have noticed that we cannot find a *diameter* of G in $O(n^2)$ time during the first step of the heuristic; the fastest known methods are $O(n^{5/2})$ [10]. In practice, we instead use a longest BFS path, starting from a random vertex. This is appropriate, by the following

Theorem: For a connected random graph G with bounded degree, the depth of BFS starting at a random node equals $diam(G) - O(1)$ with probability near 1. □

Actually, a stronger result can be proved, namely that *any* pair of vertices in G are "far" from each other with probability approaching 1 as the size of G increases.

We also have the following, due to [3]:

Theorem: The diameter of random connected graphs with bounded degree is $O(\log n)$. □

whence it may be shown that

Corollary: For a connected intersection graph G with bounded degree $\leq d$, the expected size of the boundary set, $|B|$, is cn, where c is a constant. □

So, partition quality does not vary with size of the input hypergraph.

The r-bipartition Constraint.

Finally, we mention the weighted r-bipartition constraint, which has been ignored thus far due to its highly practical nature. In addition, as noted above, many practical VLSI applications do not require enforcement of the equipartition constraint. It can also be shown that the basic algorithm results in near-equal weight partition with high probability.

To enforce weight equipartition, we begin by examining the boundary graph. Expected boundary set size implies that the equipartition of total vertex weight (i.e., module area in the VLSI paradigm) will almost never be violated by the initial cut. We therefore need worry about weight equipartition only when processing the bipartite boundary graph G'. Define the weight of a side of the partial G bipartition to be the sum of the weights of all H-vertices belonging to non-boundary and winner nodes on that side. To perform the set partitioning, we use an $O(n \log n)$ heuristic rule analogous to the so-called "engineer's method".

Rule: If the left (right) side of the partition has less weight than the right (left), pick the smallest-degree vertex remaining in G'_L (G'_R) as the next winner.

This method results in a very balanced weight partition, with maximum error dependent on d, r, etc. Note that the heuristic can also be extended to accomodate weighted edges in G. In practice, we find that the improved weight partition is obtained at the cost of slightly higher cutsizes, much as one would suspect. It would be interesting to study the utility of this "engineer's" heuristic with respect to such "balance-oriented" metrics as the minimum quotient cut.

4. Results

Algorithm I was implemented and tested on a wide range of sample industry VLSI/PCB netlists as well as difficult random inputs. Typical results for Algorithm I, simulated annealing and min-cut are shown in Table 2, where cutsizes have been normalized to 100.0 and CPU entries denote the average of ratios of runtimes, taken over all instances. These results indicate that Algorithm I is as good as, or better than, methods in [4][8][18].

Example (Mods,Sigs)	Alg I Cutsize	SA Cutsize	MinCut-KL Cutsize
Bd1 (103,211)	100.0	103.5	98.4
Bd2 (619,1527)	100.0	105.1	113.5
Bd3 (242,502)	100.0	97.0	119.9
IC1 (561,800)	100.0	109.6	109.9
IC2 (2471,3496)	100.0	88.5	98.0
Diff1 (500,700)	100.0	812	1542
Diff2 (500,700)	100.0	630	1270
Diff3 (500,700)	100.0	1216	833
CPU	1.0	$> 10^3$	120

Table 2

For difficult examples with bounded d and r, and with optimum cutsize of $o(n^{1/d})$, Algorithm I always found a min-cut bipartition, while Kernighan-Lin and annealing methods often became stuck at a terrible bipartition. For completely pathological cases where $c = 0$, BFS in G finds the unconnectedness while standard heuristics will often output a locally minimum cut of size $\Theta(|E|)$. In practice, Algorithm I is significantly faster than all existing heuristics.

A final observation: our example netlists typically have intersection graph diameter greater than that of random hypergraphs with similar degree sequences. We suspect that this is due to natural functional partitions (logical hierarchy) within the netlist. This perhaps implies that our partitioning method is even better suited to circuit designs than to random hypergraphs.

Extensions.

Because the algorithm is so fast, a natural extension of our method involves examining more than one initial longest path in G. The test runs reported below examined 50 random longest paths and selected the best result.

Another extension we are investigating involves netlist *granularization* by replacing larger modules with linked uniform small modules. This seems to work particularly well in the standard-cell regime, where cell area is roughly proportional to the number of I/Os. Our experiments are incomplete, but it seems that the weight bipartition is more balanced.

It is also interesting to consider alternative greedy methods for partitioning the boundary graph G', proving error bounds as appropriate. For example we have found success with several variants of the *Complete_Cut* method above. Finally, we are examining the performance of Algorithm I for different metrics, especially the quotient cut.

5. Conclusions

In conclusion, we have introduced a fast heuristic for hypergraph min-cut partitioning in VLSI placement. The method is extremely simple, yet provably good. The theoretical complexity bound is $O(n^2)$, and tests verify this execution speed. Results for circuit partitioning in a wide range of technologies were very encouraging. For randomly generated "difficult" hypergraphs the performance is almost always optimum.

6. Acknowledgements

This work was supported in part by NSF Grant MIP-8700767, by a grant from the State of California MICRO program, and by a Powell Foundation Fellowship. The author wishes to thank Professor T. C. Hu for much advice and encouragement.

7. References

1. Berge, C., *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.
2. Bollobas, B., *Random Graphs*, Academic Press, 1985.
3. Bollobas, B. and W. F. de la Vega, "Diameter of Random Regular Graphs", *Combinatorica* 2(4), 1982, pp. 125-134.
4. Breuer, M. A., "Min-Cut Placement", *J. Design Automation and Fault Tolerant Computing* 1(4), October 1977, pp. 343-362.
5. Bui, T.N., S. Chaudhuri, F. T. Leighton and M. Sipser, "Graph Bisection Algorithms With Good Average Case Behavior", *Combinatorica* 7(2), 1987, pp. 171-191.
6. H. W. Carter, M. A. Breuer and Z. A. Syed, "Incremental Processing Applied to Steinberg's Placement Procedure", *Proc. 12th Design Automation Workshop*, 1975, pp. 26-31.
7. Chopra, S., "Hypergraph Partitioning and VLSI Circuits", *Technical Report*, NYU School of Business Administration, 1987.
8. A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits", *IEEE Trans. on CAD ICAS* CAD-4(1), 1985.
9. C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions", *Proc. 19th Design Automation Conference*, 1982, pp. 175-181.
10. Fredman, M., "New Bounds on the Complexity of the Shortest Path Problem", *SIAM J. Computing* 5, March 1976, pp. 83-89.
11. Fukunaga, K., S. Yamada, H. S. Stone and T. Kasai, "Placement of Circuit Modules Using a Graph Space Approach", *Proc. 20th Design Automation Conference*, 1983, pp. 465-471.
12. M. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.
13. G. Hachtel, A. R. Newton and A. Sangiovanni-Vincentelli, "An Algorithm for Optimal PLA Folding", *IEEE Trans. on CAD ICAS* CAD-1(2) (1982), pp. 63-77.
14. M. Hanan, P. K. Wolff, and B. Agule, "Some Experimental Results on Placement Techniques", *Proc. 13th Design Automation Conference*, 1976, pp. 214-224.
15. M. Hartoog, "Analysis of Placement Procedures for VLSI Standard Cell Layout", *Proc. 23rd Design Automation Conference*, 1986, pp. 314-319.
16. T. C. Hu and K. Moerder, "Multiterminal Flows in a Hypergraph", in *VLSI Circuit Layout: Theory and Design*, T.C. Hu and E.S. Kuh, Eds., IEEE Press, 1985, pp. 87-93.
17. Kernighan, B. W. and S. Lin, "An Efficient Procedure for Partitioning Graphs", *Bell System Technical Journal* 49(2), 1961, pp. 291-307.
18. Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing", *Science* 220(4598), 13 May 1983, pp. 671-680.
19. Lawler, E., *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart & Winston, New York, 1976.
20. T. Leighton and S. Rao, "An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems With Applications to Approximation Algorithms", *Proc. IEEE Symp. on Foundations of Computer Science*, May 1988, pp. 422-431.
21. J. Schmidt-Pruzan and E. Shamir, "Component Structure in the Evolution of Random Hypergraphs", *Combinatorica* 5(1), 1985, pp. 81-94.
22. Schweikert, D.G. and B.W. Kernighan, "A Proper Model for the Partitioning of Electrical Circuits", *Proc. 9th Design Automation Workshop*, 1972.
23. Soukup, J. "Circuit Layout", *Proc. of the IEEE* 69(10), October 1981, pp. 1281-1304.