Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

NET PARTITIONS YIELD BETTER MODULE PARTITIONS

J. Cong
L. Hagen
A. Kahng

November 1991
CSD-910075

Net Partitions Yield Better Module Partitions

Jason Cong, Lars Hagen and Andrew Kahng

UCLA Department of Computer Science
Los Angeles, CA 90024-1596

Abstract

Partitioning of circuit netlists is important in many phases of VLSI design, ranging from layout to testing and hardware simulation. Most partitioning formulations are intractable, and the problem is further complicated in part because logic circuits are hypergraphs, not graphs. Thus, various standard net models (e.g., a k-clique or a spanning k-tree for a k-pin net) have been used to convert the netlist hypergraph into a graph representation. In this paper, we demonstrate that the "dual" intersection graph of the hypergraph better captures netlist properties relevant to partitioning. We apply this transformation within a testbed that uses an eigenvector computation to optimize the ratio cut metric of Wei and Cheng [32]; the eigenvector approach has previously been shown by Hagen and Kahng [13] to be more effective than standard Kernighan-Lin based methods, particularly for ratio cut optimization. The eigenvector computation yields a linear ordering of *nets*, rather than modules; we compute a good module partition via a sequence of incremental independent-set computations in bipartite graphs that are derived from the net ordering. An efficient matching-based algorithm called IG-Match was tested on MCNC benchmark circuits as well as additional industry examples. Results are quite promising: the algorithm yields an *average* of 28.8% improvement over the results of [32]. The intersection graph representation also yields speedups over, e.g., the method of [13], due to additional sparsity in the netlist representation.

1 Preliminaries

With increasing system complexity, a hierarchical divide-and-conquer approach is used to keep the layout synthesis process tractable. Since early design decisions will constrain all succeeding decisions, the high-level phases are critical to the quality of the final layout. Many good solutions to the placement, global routing and detailed routing problems depend on the output of the partitioning algorithm. This is especially true with the growing emphasis on performance issues in layout: signal delays typically decrease as one moves downward in the design hierarchy, e.g., on-chip communication is faster than inter-chip communication, and so the foremost goal in decomposition is to minimize the number of (critical) signal nets which cross between subproblems. This partitioning formulation is basic to many CAD applications, including wireability analysis in synthesis, the packaging or repackaging of designs, and clustering analysis for, e.g., floorplanning [5]. In addition, Wei and Cheng [32] note that partitioning can be crucial to efficient hardware simulation and test: a good partitioning will minimize the number of

signals between blocks that are multiplexed onto a hardware simulator; similarly, reducing the number of inputs to a block implies that fewer vectors will be needed to exercise the logic.

1.1 Previous Partitioning Formulations

A standard model for VLSI layout associates a graph $G = (V, E)$ with the circuit netlist; vertices in V represent modules and edges in E represent signal nets. The vertices and edges of G may be weighted to reflect module area and the multiplicity or importance of a wiring connection. Because nets often have more than two pins, the netlist is more generally represented by a *hypergraph* $H = (V, E')$, where hyperedges in E' are the subsets of V contained by each net [27]. A large portion of the literature has treated graph partitioning instead of hypergraph partitioning: not only is the formulation simpler, but many algorithms are applicable only to graph instances.

Two basic (graph) formulations for circuit partitioning are:

- **Minimum Cut:** Given $G = (V, E)$, find the *min-cut* partition of V into disjoint U and W such that $e(U, W)$, i.e., the number of edges in $\{(u, w) \in E \mid u \in U \text{ and } w \in W\}$, is minimized.
- **Minimum-Width Bisection:** Given $G = (V, E)$, find the partition of V into disjoint U and W, with $|U| = |W|$, such that $e(U, W)$ is minimized.

Although a minimum cut may be found in polynomial time due to the max-flow min-cut theorem of Ford and Fulkerson [8], the cut will often divide modules very unevenly. Because minimum-width bisection divides module area evenly, it has been a more popular objective, particularly within hierarchical approaches. However, the area bisection requirement is unnecessarily restrictive, and various ad hoc thresholds and penalty functions (e.g., the r-bipartition formulation of Fiduccia and Mattheyses [7]) have been used with varying degrees of success to relax this constraint. With this in mind, the recent *ratio cut* metric of Wei and Cheng [32] has proved to be a highly successful objective function.

- **Minimum Ratio Cut:** Given $G = (V, E)$, find the partition of V into disjoint U and W such that $\frac{e(U,W)}{|U| \cdot |W|}$ is minimized.

The ratio cut metric intuitively allows freedom to find "natural" partitions: the numerator captures the minimum-cut criterion, while the denominator favors an even partition. A good ratio cut is very useful: [32] reports average cost improvements of 39% over previous standard methods [7] on industry

benchmarks. Additionally, for testability and hardware simulation applications the recent Ph.D. thesis of Wei [33] and [34] report extraordinary cost savings of up to 70% in a number of industry settings. Unfortunately, finding either a minimum-width bisection or a minimum ratio cut is NP-complete [11] (the latter by reduction from Bounded Min-Cut Graph Partition), so heuristic methods must be used. Previous approaches to min-width bisection fall into several classes, as surveyed in [5] [14] [22]. Most of these approaches can also be applied to the minimum ratio cut objective.

In practice, iterative methods are popular either as stand-alone strategies or as a postprocessing refinement to other methods. Iterative methods are based on local perturbation of a current solution and can be greedy (the Kernighan-Lin method [19] [27] and its algorithmic speedups by Fiduccia and Mattheyses [7] and Krishnamurthy [21]), or stochastic (the hill-climbing "annealing" approach of Kirkpatrick et al. [20], Sechen [28], and others). Practical implementations will use a number of random starting configurations and return the best result [22] [32] in order to adequately search the solution space and give predictable performance, or "stability". For example, Wei and Cheng [32] use an adaptation of the shifting and group swapping methods in [7] in order to achieve the results cited above.

With respect to the present work, the most important class of partitioning algorithms consists of "spectral" methods which use eigenvalues or eigenvectors of matrices that are derived from the netlist graph. Recall that the circuit netlist may be represented by the simple undirected graph $G = (V, E)$ with $|V| = n$ vertices $v_1, \ldots, v_n$. Often, we use the $n \times n$ *adjacency matrix* $A = A(G)$, where $A_{ij} = 1$ if $\{v_i, v_j\} \in E$ and $A_{ij} = 0$ otherwise. If G has weighted edges, then A_{ij} is equal to the weight of $\{v_i, v_j\} \in E$, and by convention $A_{ii} = 0$ for all $i = 1, \ldots, n$. If we let $d(v_i)$ denote the degree of node v_i (i.e., the sum of the weights of all edges incident to v_i), we obtain the $n \times n$ *diagonal matrix* D defined by $D_{ii} = d(v_i)$. (When no confusion may arise, we may also use d_i to denote $d(v_i)$.) The eigenvalues and eigenvectors of such matrices are the subject of the relatively recent subfield of graph theory dealing with *graph spectra* [4].

Early theoretical work connecting graph spectra and partitioning is due to Barnes, Donath and Hoffman [1] [5] [6]. More recent eigenvector and eigenvalue methods have dealt with both module placement (Frankle and Karp [9] and Tsay and Kuh [30]) and graph min-cut bisection (Boppana [2]). In general, these previous works formulate the partitioning problem as the assignment or placement of nodes into bounded-size clusters or chip locations. The problem is then transformed into a quadratic optimization, and Lagrangian relaxation is used to derive an eigenvector formulation. (Appendix A summarizes a prototypical eigenvector formulation, which is due to Hall [15].) In [13], Hagen and Kahng established a close relationship between the optimal ratio cut cost and the second-smallest eigenvalue

of the matrix $Q = D - A$, where D and A are as defined above:

Theorem 1 (Hagen-Kahng): Given a netlist graph $G = (V, E)$ with adjacency matrix A, diagonal degree matrix D, and $|V| = n$, the second smallest eigenvalue λ of $Q = D - A$ yields a lower bound on the cost c of the optimal ratio cut partition, with $c \geq \frac{\lambda}{n}$. □

This result suggests that the eigenvector x corresponding to λ, i.e., the solution of the matrix equation $Qx = \lambda x$, be used to guide the partitioning. In [13], x was used to induce a linear ordering of the modules, and the best "split" in terms of ratio cut cost was returned. To be more specific, the n components x_i of the eigenvector were sorted, yielding an ordering $v = v_1, \ldots, v_n$ of the modules. The splitting rank r, $1 \leq r \leq n-1$, was then found which gave the best ratio cut cost when modules with rank $> r$ were placed in U and modules with rank $\leq r$ were placed in W. This straightforward construction achieved a very significant 17% ratio cut improvement over the RCut1.0 program of Wei and Cheng [32] for the MCNC Primary benchmarks [13], and a 9% average improvement over RCut1.0 for both the MCNC Primary and Test benchmarks [14]. It was shown that the spectral approach for ratio cut partitioning exhibited several desirable traits, including speed, provability, and stability. For this reason, and because the spectral approach significantly outperforms iterative Fiduccia-Mattheyses style methods, we have chosen to use eigenvector computations as the basis for our current algorithmic approach.[1]

1.2 Main Contributions

The main contribution of the present work is in pointing out advantages to using a *dual* representation of the logic design. We use statistical analyses of netlist structure and sparsity arguments to argue that *net* structure and interrelationships, rather than *module* adjacencies, should constitute the primary descriptors of a circuit. In particular, the dual *intersection graph* representation of the netlist hypergraph yields much more natural circuit partitioning formulations, since it inherently emphasizes relationships between signal nets. Moreover, the intersection graph yields a sparser circuit representation than traditional net models (for example, the MCNC Test05 intersection graph has an adjacency matrix that is over *ten times* sparser than the adjacency matrix created using the standard clique model, with 19935

[1] While eigenvalue computations are not cheap, the run-times reported in [13] were actually less than for the multiple F-M computations needed by the RCut1.0 program. Significant algorithmic speedups stem from the need to calculate only a *single* (the second-smallest) eigenvalue of a *symmetric* matrix. In particular, netlist graphs tend to be very sparse due to hierarchical circuit organization and degree bounds imposed by the technology fanout limits; this allows application of sparse numerical techniques, specifically the block Lanczos algorithm [12]. We use an existing Lanczos implementation [13] to calculate the second-*largest* eigenvalue and the corresponding eigenvector of the matrix $-Q = A - D$. (This is equivalent to computing the second-smallest eigenvector of $Q = D - A$, i.e., we compute $-\lambda$ and $-v$, and is preferable by theoretical results of Kaniel-Paige-Saad [12] which show that the Lanczos algorithm converges faster to the largest eigenvalues.)

nonzeros versus 219811 nonzeros); this allows speedup of numerical computations. Finally, the intersection graph representation is not sensitive to choice of net models since only the interrelationships between nets are studied.

When we use the intersection graph representation of the netlist, we may formulate the minimum-cost module partitioning as a two-stage process. In the first stage, we partition the *nets* of the design. Some modules will belong only to nets on one side of the partition; these modules can be unambiguously assigned to that side. However, other modules may belong to nets on both sides of the partition. Thus, the second stage of the module partitioning involves finding the best *completion* of the net partition, i.e., an assignment of each shared module to one side or the other such that the partition cost is minimized. We propose an efficient algorithm, called IG-Match, for completing the net partition; the algorithm is so named because it is based on a matching computation in a special bipartite graph. IG-Match affords a tight bound on the number of nets cut in completing any given partition; this bound is essentially best-possible. When we move from one net partition to another based on a shift in the splitting rank of the sorted eigenvector, the corresponding change in the bipartite graph is very small. Therefore, an interesting incremental strategy is possible, and the computational burden of examining all splits of the eigenvector can be effectively amortized. Empirical results are very encouraging. The IG-Match method yields significant improvements over the previous ratio-cut partitioning methods: results are an average of 28.8% better than those of Wei and Cheng [32] [34], and also give a 7% average improvement over a "voting" scheme recently proposed by Hagen and Kahng [14].

The remainder of this paper is organized as follows. In Section 2, we discuss netlist representations and define the netlist intersection graph. In Section 3, we formulate the problem of completing the module partition from a given net partition as a maximum independent set instance in a bipartite graph, and then present the IG-Match algorithm along with analyses of its performance and complexity. Section 4 gives performance results on benchmarks from MCNC and industry sources; we also give comparisons with previous work, including the RCut1.0 program of Wei and Cheng [32] and the "voting" method of Hagen and Kahng [14]. Finally, Section 5 gives conclusions and directions for future work.

2 Netlist Representations and the Intersection Graph

2.1 Standard Net Models

The graph representation of the netlist hypergraph is accomplished using a *net model* which determines the A_{ij} values in the weighted graph representation of the design. Many net models have been proposed,

including spanning paths, spanning cycles, spanning trees, star topologies, etc. Several models can suffer from nondeterministic asymmetry in the connection weights A_{ij}, i.e., not all adjacencies derived from a given k-pin net will be accorded the same significance. Furthermore, some topologies (e.g., minimum spanning tree, centroid-based star [29], etc.) are inherently dynamic, requiring recomputation with every change in the module placement (see [22] for a survey).

The most common net model is that of a weighted clique, where a k-pin net will induce $C(k,2)$ edges among its k modules. Recent work has widely adopted a "standard" weighted clique model [22], wherein a k-pin net contributes $\frac{1}{k-1}$ to each of $C(k,2)$ A_{ij} values. Obvious advantages of the clique model stem from its "fairness" (i.e., symmetry). However, the model also has a number of disadvantages. As noted by Yeh et al. [35], multi-pin net models have always presented a difficulty for iterative partitioning approaches, since the net model must somehow allow us to correctly capture both the immediate and the potential cut gains associated with any perturbation of the current module partition. Slight changes in the net model will result in significantly different output, and thus the clique representation exhibits some fragility. For spectral heuristics, the primary disadvantage of the clique model is that the resulting adjacency matrix has too many nonzeros (e.g., a 100-pin clock net will generate 4950 nonzeros), negating the effectiveness of such sparse operator methods as the Lanczos technique.

Given the fragility of the clique model and its unsuitability to sparse numerical methods, our initial investigations centered on alternate net representations. In particular, empirical analysis of the Fiduccia-Mattheyses approach led to the discovery of interesting relationships between the size of a net and the probability that the net is cut in the heuristic (ratio-cut or min-width bisection) circuit partition.

2.2 The Intersection Graph

We begin this section with a simple thought experiment [14]: Given a 2-pin net and a 14-pin net in a circuit netlist, which is more likely to be cut in the optimal ratio-cut partition? A simple random model would indicate that it is much less likely for all 14 modules of the larger net to be on a single side of the partition than it is for both modules of the smaller net to be on a single side of the partition. The 14-pin net is thus much more likely to be cut, and one might guess that that the *cut probability* for a k-pin net, given a random partition, would be roughly $1 - O(2^{-k})$. This rough relationship has indeed been confirmed for heuristic minimum-width bisections of various small netlists from industry and academia, including, e.g., ILLIAC IV printed circuit boards. However, our analysis of Fiduccia-Mattheyses and spectral output [32] [13] for both minimum-width bisection and minimum ratio-cut metrics has shown that this intuitive model does not necessarily remain correct, particularly as circuit sizes grow large.

For example, a typical locally minimum ratio cut (i.e., an optimized partition) for the MCNC Primary2 netlist yields the following statistics, shown in Table 1.

Net Size	Number of Nets	Number Cut
2	1835	21
3	365	29
4	203	18
5	192	26
6	120	5
7	52	12
8	14	0
9	83	5
10	14	1
11	35	0
12	5	0
13	3	0
14	10	0
15	3	0
16	1	0
17	72	22
18	1	1
23	1	0
26	1	1
29	1	0
30	1	0
31	1	0
33	14	4
34	1	0
37	1	0

Table 1: Cut statistics for k-pin nets. Note that the probability of a net being cut in the best heuristic partition does not necessarily increase monotonically with net size, counter to intuition.

Such statistics as these are not surprising in retrospect: while a random model may suffice for small circuits, larger netlists have strong hierarchical organization reflecting the high-level functional partitioning imposed by the designer. Thus, nets themselves may very well contain "useful" partitioning information.[2] Furthermore, if we consider the partitioning problem from a slightly different perspective, we realize that the minimum (ratio) cut metric is not only asking for an assignment of *modules* to the two sides of the partition, but is equivalently asking us to assign *nets* to the two sides of the partition, with the objective of maximizing the number of nets that are *not* cut by the partition. In other words, we want to assign the greatest possible number of nets *completely* to one side or the other of the partition. A central observation of this paper is that such an objective can be captured using the graph dual of the netlist hypergraph, also known as the *intersection graph* of the hypergraph.

The dualization of the problem is as follows. Given a netlist hypergraph $H = (V', E')$ with $|V'| = n$ and $|E'| = m$, we consider the graph $G' = (V, E_{G'})$ which has $|V| = m$, i.e., G' has m vertices, one for each hyperedge of H (that is to say, each signal in the netlist). Two vertices of G' are adjacent if and only if the corresponding hyperedges in H have at least one module in common. G' is called

[2]Interestingly, this suggests that standard thresholding methods for sparsifying the input, i.e., by disregarding large nets, may actually be discarding useful partitioning information.

the *intersection graph* of the hypergraph H. For any given H, the intersection graph G' is uniquely determined; however, there is no unique reverse construction. An example of the intersection graph is shown in Figure 1.

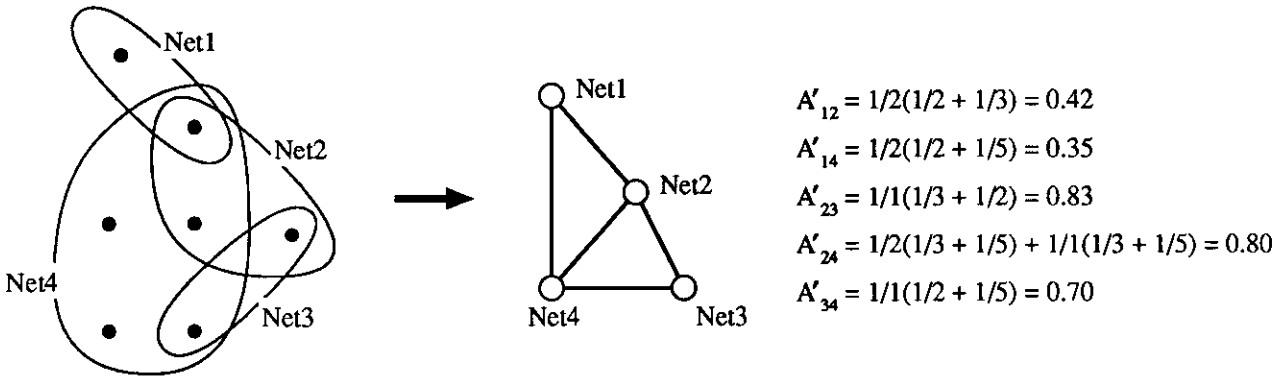

Figure 1: Left: the hypergraph for a netlist with six signal nets (each node represents a module). Right: the intersection graph of the hypergraph (each node represents a signal net). The intersection graph edge weights A'_{ij} are also shown.

Given this definition, the adjacency matrix A' of the intersection graph G' has nonzero elements A'_{ab} exactly when signal nets s_a and s_b share at least one module. As with the usual mapping of the netlist hypergraph to a graph via the weighted clique net model (Section 2.1), there are a number of possible heuristic edge weighting methods for the intersection graph. We have tried several approaches, most of which lead to extremely similar, high-quality partitioning results; this seems to support the conclusion that the intersection graph is indeed a highly robust, natural representation. In the discussion below, we use the following weighting in the intersection graph construction:

For each pair of signal nets s_a and s_b with $q \geq 1$ nodes $v_1, \ldots, v_q$ in common, let $|s_a|$ and $|s_b|$ be the number of nodes in s_a and s_b respectively. The element A'_{ab} is then given by

$$A'_{ab} = \sum_{k=1}^{q} \frac{1}{(d_k - 1)} \left(\frac{1}{|s_a|} + \frac{1}{|s_b|} \right)$$

where d_k is again equal to the degree of the k^{th} common node v_k, i.e., the number of nets incident to module k (see Figure 1).

This net weighting scheme is designed so that overlaps between large nets are accorded somewhat lower significance than overlaps between small nets. The diagonal degree matrix D' is constructed analogously to the matrix D described in Section 1.1 above, with the D'_{jj} entry equal to the sum of the entries in the j^{th} row of A'. Thus, D'_{jj} indicates the total strength of connections between signal net s_j

and all other nets which share at least one module with s_j, i.e.,

$$D'_{jj} = \sum_{i=1}^{m} A'_{ij}.$$

Given A' and D', we then find the eigenvector x' corresponding to the second eigenvalue λ' of $Q' = D' - A'$, using the same Lanczos code as in [13]. As described in Section 1 above, the sorted eigenvector yields an ordering v' of the *net* indices, and we use this ordering to derive a heuristic *module* partition. Before presenting our new partitioning algorithm, we note that the intersection graph has had only limited previous application in the CAD literature. Pillage and Rohrer [24] applied the "nets-as-points metric" to module placement, the idea being that a heuristic 2-D placement of nets would establish preferred regions for each module – i.e., a module would wish to lie somewhere within the convex hull of the locations of nets to which it belonged). This formulation required a number of ad hoc decisions and an iterative solution scheme, mostly because the intersection graph is not naturally suited to placement. For partitioning, Kahng [18] used diameters of the intersection graph to yield an approximate hypergraph bisection heuristic; more recently, Yeh et al. [35] proposed to compute gains from a "net perspective" in an iterative multiway partitioning approach. Finally, Hagen and Kahng [14] have recently discussed a simple spectral approach involving the intersection graph; we refer to their method in the next section as the "IG-Vote" method, for purposes of comparison with the present results.

3 Spectral Partitioning Based on the Intersection Graph

Recall that sorting the second eigenvector of the netlist intersection graph yields a linear ordering v' for the signal nets of the original netlist. Nets at one end of the sorted eigenvector will usually have very weak connections to nets at the other end, so our basic strategy is to test all possible splitting points of the linear ordering, in order to see which splits might lead to a good module partition. Although such an approach seems computationally expensive, we shall show later that an efficient incremental method can be applied.

Consider what happens when we split the vertices of the intersection graph into two sets L and R. It is possible that the sets of modules contained respectively by the nets of L and the nets of R are disjoint, and that the partition is "perfect", with net-cut zero. However, this is unlikely. Rather, a net $l_i \in L$ might share one or more modules with a net $r_j \in R$. If we draw an edge $\{l_i, r_j\}$ between all pairs of signal nets (l_i, r_j) which are on opposite sides of the split and which have at least one module

in common, we induce a bipartite graph $B(L, R, E_B)$ (see Figure 2). Note that by such results as Theorem One [13], use of the eigenvector-based ordering suggests that $|E_B|$, i.e., the number of edges in $\{(l_i, r_j)|l_i \in L, r_j \in R\}$, will be small.

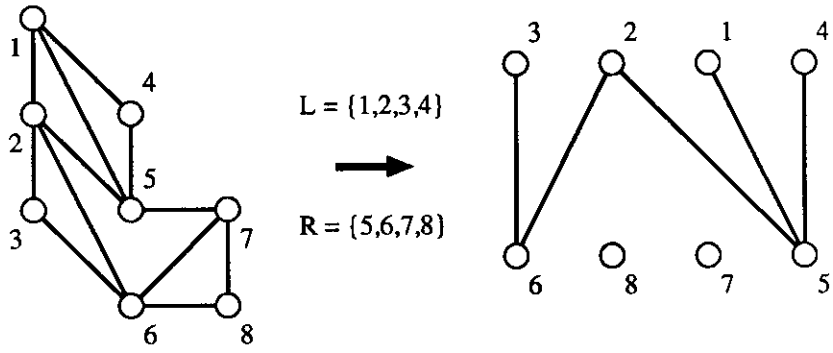

Figure 2: Inducing a bipartite graph from the intersection graph.

If a given edge $\{l_i, r_j\}$ is present in B, the key observation is that for any module partition, at most one of the following can be true: (i) l_i has all of its modules on the L-side of the partition, or (ii) r_j has all of its modules on the R-side of the partition. This follows immediately from the fact that l_i and r_j have some module in common. In [18], nets left uncut by the final module partition were called *winners*, and those cut were called *losers*. Adopting this terminology, we see that the min-cut objective is to maximize the number of winner nets, or equivalently, to minimize the number of loser nets.

In formalizing this optimization, the following graph-theoretic terms are useful.

Definition: Given a graph $G = (V, E)$, an *independent set* in G is a subset $V' \subseteq V$ such that no two nodes of V' are connected by an edge. A *maximum independent set* (MIS) is an independent set with largest possible cardinality.

Definition: Given a graph $G = (V, E)$, a *vertex cover* (VC) of G is a subset $V' \subseteq V$ such that for every edge $\{v_i, v_j\} \in E$, either $v_i \in V'$ or $v_j \in V'$. A *minimum vertex cover* (MVC) is a vertex cover with smallest possible cardinality.

Definition: Given a graph $G = (V, E)$, a *matching* in G is a set of k edges in E, no two of which have a vertex in common; we say that k is the *size* of the matching. A *maximum matching* (MM) is a matching with largest possible size.

Using these terms, the problem of maximizing the number of winners (minimizing the number of cut nets) is equivalent to finding a maximum independent set in B. While the MIS problem is NP-complete in general, it is efficiently solved for bipartite graphs. The following two standard results (see, e.g.,

Theorems 10.1 and 10.2 in [16]) motivate our algorithmic approach.

Theorem 2: For a bipartite graph $B = (L, R, E_B)$ with $|L| + |R| = n$, the sizes of any minimum vertex cover and any maximum independent set sum to n. Moreover, the complement of any MIS will be a MVC, i.e., $(L \cup R) - MIS = MVC$. □

Theorem 3: For a bipartite graph $B = (V, E)$, the size of a minimum vertex cover of B is equal to the size of a maximum matching in B. □

Given a minimum vertex cover, by Theorem 2 we may simply take its complement to obtain a maximum independent set, and vice versa. (Our plan will be to derive an MIS and make all of the corresponding signal nets winners.) Theorem 3 provides a lower bound for the size of the vertex cover (i.e., the set of loser nets); if we can find a vertex cover whose size is equal to that of the maximum matching in B, we claim optimality of the solution in that a completion has been found with net-cut as small as can be expected.

Our high-level strategy is as follows. Given the intersection graph $G' = (V, E_{G'})$, we will split the eigenvector-based net ordering v' at some index r. Placing v'_i, $i \leq r$, in L, and v'_j, $j > r$, in R, induces a bipartite subgraph $B = (L, R, E_B)$ of G'. The algorithm will try all splitting ranks $r = 1, \ldots, n-1$, where $n = |V|$. This is shown as the IG-Match main loop in Figures 5 and 6.

For each bipartite subgraph B, Phase I of the IG-Match main loop involves finding a maximum independent set in B (the reader is referred to Figure 3). We first find a maximum matching in B using the standard augmenting-path technique [23] and breadth-first search.[3] The size of the MM gives the size of the MVC, which is the number of cut nets that we hope to achieve. From the maximum matching, we construct a maximum independent set in B (i.e., the set of winner nets), as follows. Any unmatched vertex of B is a winner (Figure 3 shows the unmatched vertices on each side of the partition as U_L and U_R). Starting from any vertex in U_L or U_R, we trace alternating paths, none of which will be an augmenting path since the matching was maximum. We mark the second, fourth, etc. vertices in each of these alternating paths as losers; the (first,) third, fifth, etc. vertices in each path are marked as winners. We do this because the vertex cover (i.e., set of losers) must contain at least one vertex from every edge; in particular, it must contain at least one node from every edge in the set of alternating paths. In Figure 3, winners on paths starting at vertices of U_L are denoted as the set $Even(L)$ since

[3] Given a matching M, an *alternating path* in M consists of a path of edges $e_i \in E_B$ such that for any two consecutive edges in the path, exactly one is in M. An *augmenting path* of M consists of $2m+1$ edges $e_i \in E_B$, such that $e_i \notin M$ for i odd, $e_i \in M$ for i even, and e_i is adjacent to e_{i+1} for all $i = 1, \ldots, 2m$. The idea is that a new matching M', $|M'| = |M| + 1$, can be constructed from M by replacing the even edges of the augmenting path by the odd edges. It is well-known that a matching M is maximum if no augmenting path of M can be found.

they are at even distance from U_L, and losers on these paths are denoted as $Odd(L)$); $Even(R)$ and $Odd(R)$) are similarly denoted. Note that $U_L \subseteq Even(L)$ and $U_R \subseteq Even(R)$.

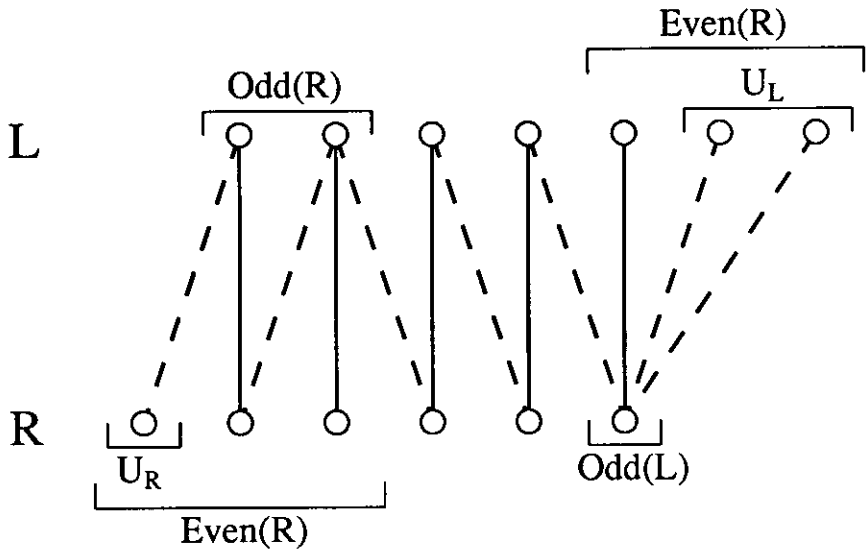

Figure 3: An example bipartite graph showing the matching M and the sets U_L, U_R, $Even(L)$, $Even(R)$, $Odd(L)$, and $Odd(R)$.

It is possible that after Phase I has been executed, there remain some edges in the matching M whose vertices do not belong to any one of the sets $Even(L)$, $Even(R)$, $Odd(L)$, or $Odd(R)$.[4] In Figure 3, these remaining vertices and their induced bipartite subgraph are denoted as $B' = (L', R', E_{B'})$. Phase II of the IG-Match main loop will consider all *modules* which have not been assigned to a side of the partition via any of the winners determined in Phase I. Essentially, Phase II will put all of these unassigned modules first on one side, then the other, and determine which option yields the better ratio cut cost. Note that this will force all nets in L' to be winners and all nets in R' to be losers, or vice versa.

We now show that the IG-Match algorithm is optimal in that the number of nets cut by the module partition will never exceed the size of the maximum matching in B.

Theorem 4: The set of loser nets output by Algorithm IG-Match is a vertex cover in B.

Proof: By the construction in Phase I of the main loop, any edge in an alternating path which began in U_L or U_R is covered by some loser net. The remaining edges in B are incident to vertices of the

[4] It is worth noting that the set of loser nets computed during Phase I of the IG-Match main loop is the so-called *critical set* described by Hasan and Liu in [17]. The critical set $C = Odd(L) \cup Odd(R)$ is the unique subset of nodes in B such that every minimum vertex cover of B contains C. Note that even though we start with an arbitrary maximum matching M in B, a result of [17] shows that we will always end up with the same Odd and $Even$, independent of which maximum matching we use.

subgraph B' defined above. Any edge between two vertices in B' is covered, since Phase II of the main loop will make losers of either all vertices in L' or all vertices in R'. The only remaining edges are those between a vertex in B' and a vertex not in B'. Without loss of generality, assume that an edge $\{a, b\}$ in this class is between vertex a in L' and vertex b in $R - R'$. If b is in $Odd(L)$, the edge $\{a, b\}$ is covered since all vertices in $Odd(L)$ are losers. And if b is in $Even(R)$, then we would have an alternating path from an unmatched vertex (in L) to a vertex in B' (i.e., a), contradicting the definition of B'. □

Theorem 5: The number of loser nets found by Algorithm IG-Match in completing the net partition is less than or equal to the size of a maximum matching in B.

Proof: No edge of the matching M can lie between a vertex in $Odd(L)$ and a vertex in $Odd(R)$, else there would be an augmenting path and M would not be a maximum matching. However, by the construction of Phase I every vertex in $Odd(L)$ and in $Odd(R)$ is incident to some edge of the matching M. By the construction of B', after Phase II has been completed, we have incorporated exactly one vertex from each matching edge into the set of losers. □

Thus, the number of losers is no more than the size of the MM, which by Theorem 3 is an optimal bound. In practice, the number of nets cut by the completed module partition can be less than the size of the MM (see the example of Figure 4). This is because a loser net v in $Odd(L)$ may in some instances only have modules in common with nets in $Even(R)$. When Phase II assigns all the modules of nets in $Even(R)$ to the R-side of the partition, the net v will end up with all its modules on the R-side and none of its modules on the L-side, i.e., net v will actually *not* be cut by the partition, even though it is a loser. Beyond the net-cut improvement achieved in Phase II, further elimination of loser nets from the cut may be possible. In view of this, an interesting extension of our algorithm would be to make recursive calls to IG-Match in order to optimally assign modules of B', B'', etc.; this is currently under investigation.

Figures 5 - 7 give detailed pseudocode for the IG-Match algorithm.

As we test all splits of the sorted eigenvector, we may retain information between the successive maximum matching computations, as well as between the successive MIS constructions. This allows efficient implementation of the IG-Match algorithm which has small amortized complexity.

Theorem 6: Given the intersection graph $G' = (V, E_{G'})$ of the netlist hypergraph, The IG-Match algorithm requires $O(|V| * (|V| + |E_{G'}|))$ time to complete the module partition for each of the $|V| - 1$ net partitions derived by splitting the sorted second eigenvector of $Q'(G')$.

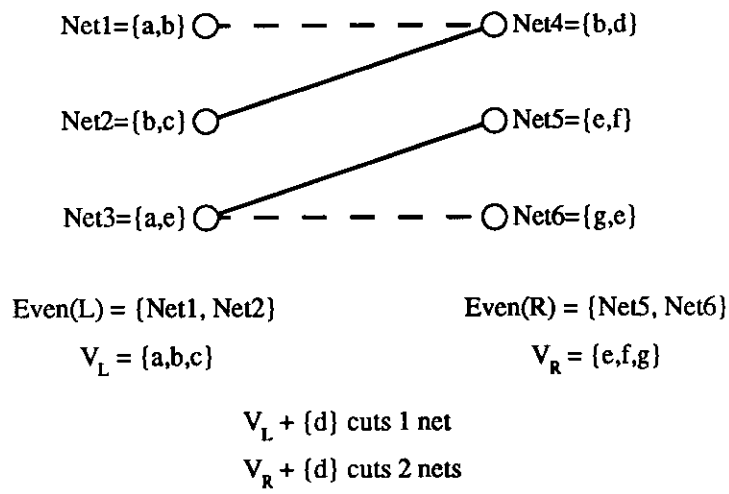

Figure 4: In the above example, the partition $V_L \cup \{d\} | V_R$ will cut fewer nets than would be indicated by the size of the maximum matching.

Proof: The breadth first searches for augmenting paths in Phase I of the main loop require time $O(|L| + |E_B|)$ to complete, while the breadth first searches to find W_L and W_R respectively require $O(|L| + |E_B|)$ and $O(|R| + |E_B|)$ time. It is clear that at any given iteration of the main loop, these time bounds are no greater than $O(|V| + |E_{G'}|)$. The remaining operations within the main loop require time $O(|V|)$ or $O(|E_{G'}|)$. Since the main loop of the algorithm is run once each time a node is moved from L to R, the complexity of the algorithm will be $O(|V| * (|V| + |E_{G'}|))$. □

4 Experimental Results

In this section, we present computational results for the IG-Match algorithm on a number of benchmark circuits from the MCNC layout test suite, as well as two additional industry circuits reported in [32]. For each benchmark, we compare our results with the *best* results of the RCut1.0 program as reported in the Ph.D. thesis of Wei [33], the earlier paper of Wei and Cheng [32], and the recent journal publication [34]. The results reported in [32] [33] are already an average of 39% better than Fiduccia-Mattheyses output in terms of the ratio cut metric; in obtaining this assessment, the authors of [32] compared the best of 10 Rcut1.0 runs to the best of 20 F-M runs, all with random starting seeds. We also compare our IG-Match results with those of an intersection graph based algorithm in [14] (the authors of [14] call their method the EIG1-IG algorithm; we use the term IG-Vote to contrast it with the present method[5]).

[5]A description of the EIG1-IG / IG-Vote method is given in Appendix B. This is included to facilitate the review process since reference [14] is not yet generally available; however, the discussion of Appendix B will not appear in the final paper.

The CPU times required by our numerical algorithms are very competitive with those cited in [32] for Fiduccia-Mattheyses optimization: for example, the eigenvector computation for PrimSC2 requires 83 seconds of CPU time on a Sun4/60, versus 204 seconds of CPU for 10 runs of RCut1.0.

While the spectral approach cannot take module areas (weights) into consideration, this has not been a significant disadvantage in practice, as witnessed by the results of Hagen and Kahng in [13]. Furthermore, that the spectral algorithm is oblivious to node weights does not present a difficulty for other large-scale partitioning applications in CAD, e.g., test or hardware simulation. For such applications, the input is simply the netlist hypergraph with uniform node weights; [33] reports that ratio cut partitioning saved 50% of hardware simulation costs of a 5-million gate circuit as part of the Very Large Scale Simulator Project at Amdahl; similar savings were obtained for test vector costs.

Test problem	Number of elements	Wei-Cheng RCut1.0			IG-Match			Percent improvement
		Areas	Nets cut	Ratio cut	Areas	Nets cut	Ratio cut	
bm1	882	9:873	1	12.73×10^{-5}	21:861	1	5.53×10^{-5}	57
19ks	2844	1011:1833	109	5.88×10^{-5}	650:2194	85	5.96×10^{-5}	-1
Prim1	833	152:681	14	1.35×10^{-4}	154:679	14	1.34×10^{-4}	1
Prim2	3014	1132:1882	123	5.77×10^{-5}	740:2274	77	4.58×10^{-5}	21
Test02	1663	372:1291	95	1.98×10^{-4}	211:1452	38	1.24×10^{-4}	37
Test03	1607	147:1460	31	14.44×10^{-5}	803:804	58	8.98×10^{-5}	38
Test04	1515	401:1114	51	11.42×10^{-5}	73:1442	6	5.70×10^{-5}	50
Test05	2595	1204:1391	110	6.57×10^{-5}	105:2490	8	3.06×10^{-5}	53
Test06	1752	145:1607	18	7.72×10^{-5}	141:1611	17	7.48×10^{-5}	3

Table 2: Output from IG-Match algorithm, compared with results from the RCut1.0 program of Wei and Cheng. Results are 28.8% better on average than those of RCut1.0.

Table 3 compares IG-Match output with computational results reported by Hagen and Kahng [14] for their IG-Vote algorithm on the same suite of test cases. We include these results in the present discussion to give additional evidence of the quality of the IG-Match algorithm, and in particular to show the effectiveness of the matching formulation. The IG-Match algorithm represents a average of 7% improvement in partition quality over the IG-Vote algorithm. As a final note, we observe that IG-Match achieves an average improvement of 22% over the original EIG1 algorithm of [13], which did not use the intersection graph representation.

5 Conclusions

We have presented a new approach to module partitioning, based on a combination of spectral techniques and the intersection graph representation G' for the circuit hypergraph. Statistical analysis of net cut probabilities as a function of net size, as well as sparsity considerations in the numerical computation,

Test problem	Number of elements	Hagen-Kahng IG-Vote (EIG1-IG)			IG-Match			Percent improvement
		Areas	Nets cut	Ratio cut	Areas	Nets cut	Ratio cut	
bm1	882	21:861	1	5.53×10^{-5}	21:861	1	5.53×10^{-5}	0
19ks	2844	662:2182	92	6.37×10^{-5}	650:2194	85	5.96×10^{-5}	7
Prim1	833	154:679	14	1.34×10^{-4}	154:679	14	1.34×10^{-4}	0
Prim2	3014	730:2284	87	5.22×10^{-5}	740:2274	77	4.58×10^{-5}	13
Test02	1663	228:1435	48	1.47×10^{-4}	211:1452	38	1.24×10^{-4}	16
Test03	1607	787:820	64	9.92×10^{-5}	803:804	58	8.98×10^{-5}	10
Test04	1515	71:1444	6	5.85×10^{-5}	73:1442	6	5.70×10^{-5}	3
Test05	2595	103:2492	8	3.12×10^{-5}	105:2490	8	3.06×10^{-5}	2
Test06	1752	143:1609	19	8.26×10^{-5}	141:1611	17	7.48×10^{-5}	10

Table 3: Output from IG-Match algorithm, compared with results from the IG-Vote algorithm of Hagen and Kahng [14]. IG-Match results uniformly dominate IG-Vote results, and IG-Match averages 7% improvement over IG-Vote.

led us to consider the intersection graph representation. We use a sparse Lanczos code to induce a linear ordering of *nets* via the sorted second eigenvector of $Q'(G')$, and formulate the *completion* of the module partition as a maximum independent set computation in a bipartite graph. Our IG-Match algorithm guarantees to complete a module partition without cutting more nets than the size of a maximum matching in the bipartite graph; this bound is tight. Furthermore, the computation is efficient in an amortized sense even when we wish to test *all* possible partitions ("splits") of the sorted eigenvector to see which leads to the best module partition: IG-Match tests all splits in $O(|V| * (|V| + |E|)$ time. Since the computational complexity of the Lanczos implementation scales well with increasing problem sizes [12], we believe that this overall methodology will continue to be useful even when problem sizes grow very large.

For the MCNC Test and Primary benchmarks, along with two additional industry benchmarks, our IG-Match algorithm obtained an average of 28.8% cost reduction over the RCut1.0 program of Wei and Cheng [32] [34]. This contrasts with the 9% improvement obtained by Hagen and Kahng [13] using the spectral approach with a traditional clique-based net model; this difference highlights the advantages of the intersection graph representation. We again note that the spectral computation is faster when we use the intersection graph, due to additional sparsity of up to an order of magnitude fewer nonzeros in the netlist representation. The IG-Match algorithm also obtained a 7% average improvement over the forthcoming "voting" heuristic of Hagen/Kahng [14]. With respect to practical advantages, our IG-Match algorithm derives its output from a single, deterministic execution of the algorithm – i.e., the approach is inherently stable and does not require multiple random starting points as with other approaches.

A number of interesting open issues remain. The eigenvector computation can be sped up further by

additionally sparsifying the input through thresholding, or by relaxation of the numerical convergence criteria. A hybrid algorithm which uses clustering to condense the input before applying the partitioning algorithm (such an approach is discussed by Bui et al. [3] and by Lengauer [22]) is also promising. Parallel speedups of the Lanczos code are also possible. With any of these heuristics, the ratio cuts so obtained may optionally be improved by using standard iterative techniques. The recursive enhancement of IG-Match described in Section 3 is also of interest. Finally, following the successes reported by Wei and Cheng [33] [34], the intersection graph based ratio cut partitioning should be applied to ratio cut partitioning for other CAD applications, particularly test and the mapping of logic for hardware simulation.

References

[1] E. R. Barnes, "An Algorithm for Partitioning the Nodes of a Graph", *SIAM J. Alg. Disc. Meth.* 3(4) (1982), pp. 541-550.

[2] R.B. Boppana, "Eigenvalues and Graph Bisection: An Average-Case Analysis", *IEEE Symp. on Foundations of Computer Science*, 1987, pp. 280-285.

[3] T. N. Bui, S. Chaudhuri, F. T. Leighton and M. Sipser, "Graph Bisection Algorithms with Good Average Case Behavior", *Combinatorica* 7(2) (1987), pp. 171-191.

[4] D. Cvetkovic, M. Doob, I. Gutman and A. Torgasev, *Recent Results in the Theory of Graph Spectra*, North-Holland, 1988.

[5] W.E. Donath, "Logic Partitioning", in *Physical Design Automation of VLSI Systems*, B. Preas and M. Lorenzetti, eds., Benjamin/Cummings, 1988, pp. 65-86.

[6] W.E. Donath and A.J. Hoffman, "Lower Bounds for the Partitioning of Graphs", *IBM J. Res. Dev.* (1973), pp. 420-425.

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

[8] L.R. Ford, Jr. and D.R. Fulkerson, *Flows in Networks*, Princeton University Press, 1962.

[9] J. Frankle and R.M. Karp, "Circuit Placement and Cost Bounds by Eigenvector Decomposition", *IEEE Intl. Conf. on Computer-Aided Design*, 1986, pp. 414-417.

[10] J. Garbers, H. J. Promel and A. Steger, "Finding Clusters in VLSI Circuits" *extended version* of paper in *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1990, pp. 520-523.

[11] M. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.

[12] G. Golub and C. Van Loan, *Matrix Computations*, Baltimore, Johns Hopkins University Press, 1983.

[13] L. Hagen and A. B. Kahng, "Fast Spectral Methods for Ratio Cut Partitioning and Clustering", to appear in *Proc. IEEE Intl. Conf. on Computer-Aided Design*, Santa Clara, November 1991.

[14] L. Hagen and A. B. Kahng, "New Spectral Methods for Ratio Cut Partitioning and Clustering", to appear in *IEEE Transactions on CAD.*

[15] K. M. Hall, "An r-dimensional Quadratic Placement Algorithm", *Management Science* 17(1970), pp. 219-229.

[16] F. Harary, *Graph Theory*, Addison-Wesley, 1969.

[17] N. Hasan and C. L. Liu, "Minimum Fault Coverage in Reconfigurable Arrays", *Proc. 18th IEEE Intl. Symp. on Fault-Tolerant Computing Systems*, 1988, pp. 348-353.

[18] A. B. Kahng, "Fast Hypergraph Partition", *Proc. ACM/IEEE Design Automation Conf.*, 1989, pp. 762-766.

[19] B.W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning of Electrical Circuits", *Bell System Technical J.*, Feb. 1970.

[20] S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi, "Optimization by Simulated Annealing", *Science* 220 (1983), pp.671-680.

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

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

[23] C. L. Liu, *Introduction to Combinatorial Mathematics*, McGraw-Hill, 1968.

[24] L. T. Pillage and R. A. Rohrer, "A Quadratic Metric with a Simple Solution Scheme for Initial Placement", *Proc. ACM/IEEE Design Automation Conf.*, 1988, pp. 324-329.

[25] A. Pothen, H. D. Simon and K. P. Liou, "Partitioning Sparse Matrices with Eigenvectors of Graphs", *SIAM J. Matrix Analysis and its Applications* 11 (1990), pp. 430-452.

[26] L.A. Sanchis, "Multiple-Way Network Partitioning", *IEEE Trans. on Computers* 38 (1989), pp. 62-81.

[27] D.G. Schweikert and B.W. Kernighan, "A Proper Model for the Partitioning of Electrical Circuits", *ACM/IEEE Design Automation Conf.*, 1972.

[28] C. Sechen, Placement and Global Routing of Integrated Circuits Using Simulated Annealing, Ph.D. Thesis, Univ. of California, Berkeley, 1986.

[29] G. Sigl, K. Doll and F. M. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?", *Proc. ACM/IEEE Design Automation Conf.*, June 1991, pp. 427-432.

[30] R.S. Tsay and E.S. Kuh, "A Unified Approach to Partitioning and Placement", *Princeton Conf. on Inf. and Comp.*, 1986.

[31] G. Vijayan, "Partitioning Logic on Graph Structures to Minimize Routing Cost", *IEEE Trans. on CAD* 9(12) (1990), pp. 1326-1334.

[32] Y.C. Wei and C.K. Cheng, "Towards Efficient Hierarchical Designs by Ratio Cut Partitioning", *IEEE Intl. Conf. on Computer-Aided Design*, 1989, pp. 298-301.

[33] Y. C. Wei, "Circuit Partitioning and Its Applications to VLSI Designs", Ph.D. Thesis, UCSD CSE Dept., September 1990.

[34] Y. C. Wei and C. K. Cheng, "Ratio Cut Partitioning for Hierarchical Designs", *IEEE Trans. on CAD*, October 1991.

[35] C. W. Yeh, C. K. Cheng and T. T. Lin, "A General Purpose Multiple Way Partitioning Algorithm", *Proc. ACM/IEEE Design Automation Conf.*, June 1991, pp. 421-426.

IG-Match for $B = (L, R, E_B)$. Main Loop Phase I: Selecting Winner Nets

L, R : sets of net-vertices in left, right partitions
E_G : set of edges in the intersection graph G'
E_B : set of edges between L and R
M : set of edges in maximum matching between L and R
P : augmenting path from R to L
N : set of net-vertices to examine in breadth first search
W_L, W_R : sets of winner net-vertices in L, R

```
v ∈ L      /* v is the next net in the sorted IG eigenvector */
L := L − {v}
for all edges (v, y) ∈ E_B do
    E_B := E_B − {(v, y)}

/* Construct a maximum matching */
if ∃u such that (v, u) ∈ M then
    M := M − {v, u}
    P := an augmenting path from u to L
    if |P| ≠ 0 then          /* augment M according to the edges in P */
        M := M − {(x, y) : x ∈ L, y ∈ R, (x, y) ∈ P}
        M := M ∪ {(x, y) : x ∈ L, y ∈ R, (y, x) ∈ P}
R := R ∪ {v}
for all edges (x, v) ∈ E_G do
    if x ∈ L then
        E_B := E_B ∪ {(x, v)}
P := an augmenting path from v to L
if |P| ≠ 0 then          /* augment M according to the edges in P */
    M := M − {(x, y) : x ∈ L, y ∈ R, (x, y) ∈ P}
    M := M ∪ {(x, y) : x ∈ L, y ∈ R, (y, x) ∈ P}

/* Construct a maximum independent set */
W_L := set of unmatched net-vertices of L, i.e., U_L
N := W_L
while N ≠ ∅ do
    let x ∈ N
    N := N − {x}
    for all edges (x, y) ∈ E_B do
        if (x', y) ∈ M and x' ∉ W_L then
            W_L := W_L + {x'}
            N := N + {x'}
endwhile /* W_L = Even(L) */
W_R := set of unmatched net-vertices of R, i.e., U_R
N := W_R
while N ≠ ∅ do
    let y ∈ N
    N := N − {y}
    for all edges (x, y) ∈ E_B do
        if (x, y') ∈ M and y' ∉ W_R then
            W_L := W_L + {y'}
            N := N + {y'}
endwhile /* W_R = Even(R) */
```

Figure 5: First phase of main loop, **Algorithm IG-Match**: selecting the winner nets.

IG-Match for $B = (L, R, E_B)$. Main Loop Phase II: Module Assignment

L, R : sets of net-vertices in left, right partitions
E_B : set of edges between L and R
W_L, W_R : sets of winner net-vertices in L, R
V_L, V_R : sets of modules contained by nets in W_L, W_R resp.
V_N : set of modules not contained by nets in W_L or W_R

```
V_L := ∅
V_R := ∅
for all nets x ∈ W_L do
    V_L := V_L ∪ { modules in net x}
for all nets y ∈ W_R do
    V_R := V_R ∪ { modules in net y}
V_N := V − (V_L ∪ V_R)
calculate ratio-cut with partitions V_L ∪ V_N and V_R
calculate ratio-cut with partitions V_L and V_R ∪ V_N
```

Figure 6: Second phase of main loop, **Algorithm IG-Match**: constructing the module partition.

Overall IG-Match Algorithm for $B = (L, R, E_B)$

L, R : sets of net-vertices in left, right partitions
E_B : set of edges between L and R
M : set of edges in maximum matching between L and R
V : set of modules in netlist

```
L := V; R := ∅; E_B := ∅; M := ∅
while L ≠ ∅ do
    Phase I : select winner nets
    Phase II : perform module assignment to partitions
endwhile
```

Figure 7: High-level outline of complete **Algorithm IG-Match.**

6 Appendix A

A prototypical example is the work of Hall [15], which we now outline. This work is particularly relevant since it uses eigenvectors of the same graph-derived matrix $Q = D - A$ (the same D and A defined above) that we utilize. Donath and Hoffman, Boppana, and others use different matrices derived from the netlist graph, but exploit similar mathematical properties (e.g., symmetry, positive-definiteness) to derive alternate eigenvalue formulations and relationships to partitioning.

Hall's result [15] was that the eigenvectors of the matrix $Q = D - A$ solve the *quadratic placement* problem of finding the vector $x = (x_1, x_2, \ldots, x_n)$ which minimizes

$$z = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i - x_j)^2 A_{ij}$$

subject to the constraint $|x| = (x^T x)^{1/2} = 1$, with A_{ij} again equal to the strength of the connection between modules i and j.

It can be shown that $z = x^T Q x$, so that to minimize z we may form the Lagrangian

$$L = x^T Q x - \lambda (x^T x - 1).$$

Taking the first partial derivative of L with respect to x and setting it equal to zero yields

$$2Qx - 2\lambda x = 0,$$

and this can rewritten as

$$(Q - \lambda I)x = 0$$

where I is the identity matrix. This is readily recognizable as an eigenvalue formulation for λ, and the eigenvectors of Q are the only nontrivial solutions for x. The minimum eigenvalue 0 gives the uninteresting solution $x = (1/\sqrt{n}, 1/\sqrt{n}, \ldots, 1/\sqrt{n})$, and hence the eigenvector corresponding to the second smallest eigenvalue λ is used.

7 Appendix B – The IG-Vote Method of [14]

Given the intersection graph G' and the net ordering v' based on the sorted second eigenvector x' of $Q'(G')$, it is too simplistic to construct the $(U|W)$ module partition by merely assigning signal net $s_{v'_1}$ to U, signal net $s_{v'_m}$ to W, signal net $s_{v'_2}$ to U, etc. This is because such assignments will soon begin to conflict: a net assigned to U will contain some module that also belongs to a net already assigned to W. To escape this difficulty, the authors in [14] adopted the following strategy: assign a module M_i to, e.g., U only when *enough* of the nets containing M_i have been assigned to U. This is accomplished with a heuristic weighting function, where each net exerts "weight" on its component modules inversely proportional to the size of the net. In practice, to guarantee that every node is assigned to a partition, all nets/modules are first placed in U, and then the nets are moved one by one to W (beginning with $s_{v'_1}$ and continuing through $s_{v'_m}$). A module will move to W only when enough of its total incident *net-weight* w_i (i.e., more than some threshold proportion) has been shifted to W. In the pseudocode below, and in the experiments reported in [14], a threshold of $\frac{1}{2} \cdot$ *net-weight* is used. Symmetrically, one may also start with all nets/nodes in W, and shift nets beginning with $s_{v'_m}$, since this yields a different set of heuristic node partitions. The output is the best ratio-cut partition among the up to $2 \cdot (n-1)$ distinct heuristic partitions so generated. The conversion of the sorted second eigenvector to a heuristic node partition is summarized in Figure 8:

Module Assignment to Partitions – IG-Vote
$\underline{w} \equiv$ array containing the *total* net weight of each node
$\underline{z} \equiv$ array containing the *moved* net weight of each node
Compute eigenvector x' of second eigenvalue $\lambda(Q'(G'))$;
Sort entries of x', yielding ordering v' of net indices;

{* *initialize net-weight vector* *}
$\underline{w} = \underline{0}$
for $i = 1$ to $n =$ number of modules
    for each signal net s_j containing module M_i
        add $1/|s_j|$ to w_i

{* *begin with all nets/nodes assigned to partition U* *}
$\underline{z} = \underline{0}$
for $j = 1$ to $m =$ number of nets
    for each module M_i in net $s_{v'_j}$
        add $1/|s_{v'_j}|$ to z_i
        if $z_i \geq (w_i/2)$
            move module M_i from partition U to partition W
    calculate and output the ratio cut cost for (U, W) partition

{* *begin with all nets/nodes assigned to partition W* *}
$\underline{z} = \underline{0}$
for $j = m$ down to 1
    for each module M_i in net $s_{v'_j}$
        add $1/|s_{v'_j}|$ to z_i
        if $z_i \geq (w_i/2)$
            move module M_i from partition W to partition U
    calculate and output the ratio cut cost for (U, W) partition

Output best ratio cut partition found.

Figure 8: High-level description of the IG-Vote heuristic [14] for module partitioning from the sorted eigenvector of the intersection graph.