Hybrid Spectral/Iterative Partitioning

Jason Y. Zien, Pak K. Chan, Martine Schlag

UCSC-CRL-97-09
April 29, 1997

Baskin Center for
Computer Engineering & Computer Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

Abstract

Although spectral partitioning has been an active area of research, there are still many limitations which prevent its widespread use. These limitations include the inability to work directly with a hypergraph model, great difficulty in specifying design constraints, and the inability to specify arbitrary cost functions. None of those limitations are present in the commonly-used Kernighan-Lin/Fiduccia-Mattheyses (KLFM) style iterative improvement heuristics. Our current work focuses on developing a new multi-way, hybrid spectral/iterative hypergraph partitioning algorithm which combines the strengths of spectral partitioners and iterative improvement algorithms to create a new class of partitioners. We show how spectral information (the eigenvectors of a graph) can be incorporated into an iterative partitioning framework. We use spectral information to generate initial partitions, influence the selection of iterative improvement moves, and break out of local minima which may trap KLFM improvement algorithms. Our 3-way and 4-way partitioning results are better than the best published results, demonstrating the effectiveness of our new hybrid method. Our hybrid algorithm produces an average improvement of 27.9% over GFM [33] for 3-way partitions, 48.7% improvement over GFM for 4-way partitions, and 67.5% improvement over ML_F [5] for 4-way partitions.

1 Introduction

1.1 The Problem

This paper examines the hypergraph partitioning problem, whose goal is to divide a large system of connected components into several smaller subsystems. A hypergraph, which is a generalization of the graph model [38], is used as an abstract representation of a more specific problem. A hypergraph is composed of vertices (nodes) having arbitrary sizes, and weighted hyperedges which connect the vertices together. The goal of hypergraph partitioning is to divide the vertices of a hypergraph into several distinct subsets subject to size or balance constraints while minimizing the interconnections among those subgraphs. Graphs are simply hypergraphs where all hyperedges are incident upon exactly two vertices.

Partitioning algorithms are useful in many areas, such as circuit placement [15] [24], minimizing communication in parallel processing simulations [42], optimizing the organization of large computer networks, and circuit implementation in field-programmable gate arrays (FPGAs) [34], [9], [12], [40]. Since partitioning with balance constraints is NP-complete [20], we must resort to heuristics to solve the problem.

1.2 Contributions

This paper describes a new multi-way, hybrid spectral/iterative, graph partitioning algorithm. We describe the theory behind our new spectral/iterative algorithm, show how to combine spectral and iterative partitioners, and present experimental results which validate our algorithm. Our methods and ideas are general enough to be applied to almost *any* current and future iterative improvement heuristics. Our contributions include:

1. A new hybrid spectral/iterative partitioning algorithm. This is the first algorithm that simultaneously combines iterative and spectral information in a multi-level partitioning framework. Previously, researchers have combined the two approaches by finding a solution using a spectral algorithm and then performing iterative improvement on it [6, 25, 44]. The simultaneous use of spectral and traditional gain costs was found in [11], where a single-pass constructive heuristic was used to create partitions using the weighted sum of the cut gain and a spectral cost function. That heuristic was only a constructive heuristic, and not used for iterative improvement. In contrast, our current work provides a much more tightly coupled integration of spectral and iterative improvement methods. The main contributions of our work include the use of circular orderings to generate multiple initial partitions, using spectral information within a Kernighan-Lin/Fiduccia-Mattheyses iterative improvement algorithm [29, 17] to break ties in gain, and using spectral information to break out of local minima which may trap standard iterative improvement algorithms.
2. A new k-way improvement method which we call Rotary KLFM.
3. The (current) best known 3-way and 4-way hypergraph partitioning results. Our hybrid algorithm produces an average improvement of 27.9% over GFM [33] for 3-way partitions, 48.7% improvement over GFM for 4-way partitions, and 67.5% improvement over ML_F [5] for 4-way partitions.

2 Background

2.1 State-of-the-Art Partitioners

The classic Kernighan-Lin/Fiduccia-Mattheyses (KLFM) algorithm even today is used as the basis for most modern iterative partitioning algorithms. This algorithm combines a greedy hill-climbing approach with a simple backtracking step, as shown in Figure 2.1. Vertex moves are selected based on the gain cost function. The gain is typically the total weight of the nets that would become uncut by moving the vertex from one partition to another. The algorithm is extremely fast, running in linear time per pass with respect to the circuit size, is fairly easy to implement, and is easily adjusted to take various cost functions or constraints into account. Improvements have been made in the selection of moves by biasing clusters of neighbors to be moved in sequence (the CLIP algorithm [16]), detecting clusters [16], and breaking ties in gain by using look-ahead [31]).

KLFM algorithms require a large number of random starts to obtain good partitioning solutions. Because KLFM algorithms are so sensitive to initial starting points, some researchers have sought ways to create more stable performance by using clustering and multiple levels of hierarchy.

A top-down approach to finding clusters in circuits was implemented by Wei and Cheng [41, 42, 13]. They used the ratio-cut partitioning algorithm to recursively subdivide a circuit into many small clusters. The ratio-cut cost function is the ratio of edges cut over the product of the partition sizes, $\frac{E_c}{|P_1|\cdot|P_2|}$. They created a contracted graph by collapsing each cluster into a supernode. They partitioned this contracted graph, then re-expanded it and ran their ratio-cut iterative improvement algorithm upon the expanded graph.

Other researchers have tried bottom-up clustering algorithms for improving results. In [19], they created a linear-time clustering algorithm to find well-connected clusters using graph connectivity properties. In [35], they performed clustering based on Rent's rule, and then executed the KLFM

```
ALGORITHM: KLFM
INPUT: hypergraph, initial 2-way partitioning solution
OUTPUT: improved 2-way partitioning solution
METHOD:
Do {
    Unlock all vertices
    Initialize vertex gains
    While (there are free vertices) {
        Select best move
        Move vertex and lock in place
        Update gains of neighbors
        If cost is best, save this position
    }
    Rewind to best position seen
} While (cost has improved)
```

Figure 2.1: The KLFM iterative improvement algorithm.

algorithm on the contracted graph. In [14], they developed a clustering algorithm by collapsing cliques into supernodes, and then running the KLFM algorithm on the contracted graph.

These methods were subsequently improved upon by using multiple levels of contraction (instead of just one level), and iterative improvement at each level of the hierarchy [26, 28, 27, 4, 44]. Current state-of-the-art partitioners use the multi-level (hierarchical) approach as shown in Figure 2.3. A hypergraph is reduced in size by pre-clustering nodes or matching nodes together (Figure 2.2). A matching of a graph is a set of edges whose vertices are non-overlapping. From this pre-clustering or matching, a contracted graph which is much smaller than the original graph is created. Figure 2.2a shows a matching of the graph. The thick edges shown are the edges selected for a matching. Figure 2.2b shows the contracted graph. The numbers inside the vertices denote the weight of the vertex, and the numbers next to the edges denote the size of the edge. Several levels of contraction can be performed to further reduce the problem size (Figure 2.3).

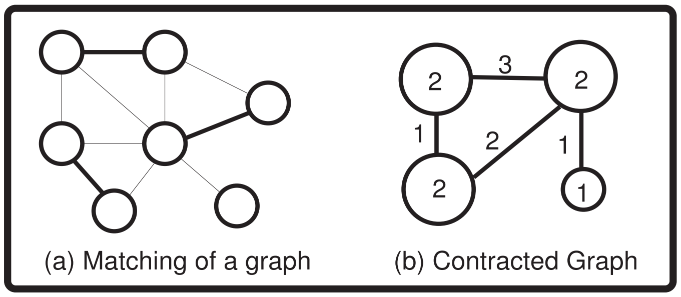

Figure 2.2: Contracting a graph by matching edges.

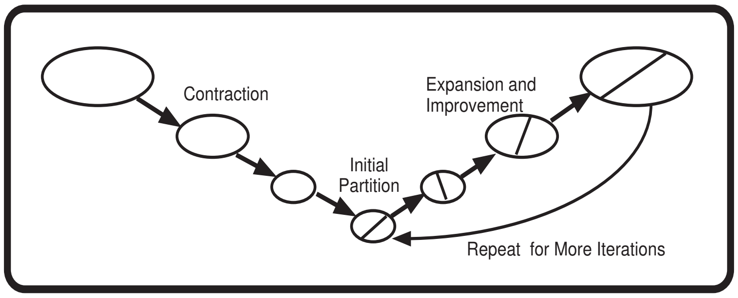

Figure 2.3: Hierarchical Partitioning

3 Spectral Partitioning and Vectors

In this section, we will give a brief overview of spectral partitioning. Since much of this theory has been published in the past, we will not repeat it here. The interested reader may refer to other papers [24, 23, 3, 18, 44] for more details.

Assume we are given a graph with n vertices, and we wish to find k partitions of this graph. Let v_i be vertex i and $deg(v_i)$ be the degree of v_i. An $n \times n$ adjacency matrix, A, is composed of entries a_{ij} which represent the weight of an edge between vertices v_i and v_j. The $n \times n$ diagonal degree matrix, D, has entries d_{ii} equal to the sum of the weights of all edges on vertex v_i. The Laplacian matrix is defined as $Q = D - A$. E_h is the total sum of the weights of the edges cut on partition h. Let M be an $n \times n$ diagonal matrix whose entries m_{ii} represent the size. S is the matrix such that $S^T S = M$.

Spectral partitioning is based upon using the eigenvectors associated with the smallest eigenvalues of the Laplacian, Q. The eigenvectors are actually the solution to a relaxed version of the partitioning problem, often referred to as the quadratic placement problem [24]. This relaxed problem optimizes the total sum of squared wiring distance of the nodes in a graph. For instance, the quadratic cost of a one-dimensional (linear) placement of nodes is $z = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2 a_{i,j}$. A spreading constraint, $x^T x = 1$ is used to spread the vertices out. The trivial solution, where all vertices are placed at a single point, corresponds to the eigenvector associated with the smallest eigenvalue, is often ignored. Many researchers have used the second eigenvector to form partitions [18, 36, 23]. A variation of this which takes vertex sizes into account was proposed in [44]. Different researchers have chosen to use different numbers of eigenvectors in creating partitions. To be general, let us assume we are using d eigenvectors, where d is the desired number of dimensions we use, so X_d is our $n \times d$ relaxed solution. There have been many attempts at using this information in different ways. For instance, some researchers have sought to find a k-way partitioning solution by finding the binary partition assignment matrix which satisfies partitioning constraints that is closest to the $n \times k$ eigenvector matrix X_k using a transportation problem [7, 8]. Another approach is to view each row of X_d as a coordinate for the vertex. The partitioning problem can then be solved by using geometric clustering algorithms [1]. Another view is to think of each row of X_d as a d-dimensional vector. Partitions can then be formed by grouping vertices together based on the angle between vectors [10] or by scaling X_d and optimizing for the maximum sum vector partitioning cost (discussed in the next section) [3].

3.1 Maximum Sum Vector Partitioning

Let X_d be the $n \times d$ matrix composed of the first d eigenvectors (those associated with the smallest d eigenvalues) of the graph and Λ_d be the $d \times d$ diagonal matrix composed of the smallest d eigenvalues of the graph. The matrix H_d is a diagonal matrix whose entries are all γ. The only constraint on the value of γ is that it must be at least as large as λ_d. In maximum sum vector partitioning, the scaled matrix of eigenvectors is represented by $V_d = MX_d\sqrt{(H_d - \Lambda_d)}$ where X_d and Λ_d satisfy $QX_d = MX_d\Lambda_d$. This is a generalization of the work of [3, 18] who used $V_d = X_d\sqrt{(H_d - \Lambda_d)}$, where X_d and Λ_d satisfy $QX_d = X_d\Lambda_d$.

Let the vector ν_i, $1 \leq i \leq n$ be row i of V_d. In maximum sum vector partitioning (MSVP), we wish to divide the vectors ν_i into k distinct sets of vectors. Let $T_h = \sum_{\nu_i \in P_h} \nu_i$ be the sum of each of those sets of vectors with $1 \leq h \leq k$. The goal of MSVP is to maximize $\sum_{h=1}^{k} ||T_h||^2$. When $d = n$, maximum sum vector partitioning is equivalent to minimizing the sum of the edges cut in graph partitioning.

The more eigenvectors being used, the closer an approximation to the graph partitioning problem it is. However, we note that there is a tradeoff in that as you use more eigenvectors, the multi-dimensional embedding becomes more and more difficult to take advantage of. In particular, when $d = n$, we have merely replaced one NP-Hard problem with another!

Throughout this chapter, however, for the purposes of establishing and proving various results, we use the full $n \times n$ matrices H, V and X.

We will derive a new proof of the maximum sum vector partitioning problem. Our proof is simpler than the proof in [3] and also more general, because it takes vertex sizes into account.

Lemma 1: *Given $\hat{X}$ which is orthonormal and H, a diagonal matrix whose entries equal the constant γ, $\hat{X}H\hat{X}^T = H$*

PROOF:

$$\begin{aligned}
\hat{X}H\hat{X}^T_{ig} &= \sum_{k=1}^{n}\sum_{j=1}^{n} (\hat{x}_{ij}h_{jk})\,\hat{x}_{kg} \\
\hat{X}H\hat{X}^T_{ig} &= \sum_{k=1}^{n}\sum_{j=1}^{n} \hat{x}_{ij}h_{jk}\hat{x}_{kg}
\end{aligned}$$

□

Since $h_{jk} = 0$ when $j \neq k$ and $h_{jj} = \gamma$:

$$\begin{aligned}
\hat{X}H\hat{X}^T_{ig} &= \sum_{j=1}^{n} \hat{x}_{ij}h_{jj}\hat{x}_{jg} \\
\hat{X}H\hat{X}^T_{ig} &= \gamma\sum_{j=1}^{n} \hat{x}_{ij}\hat{x}_{jg} \\
\hat{X}H\hat{X}^T_{ig} &= \begin{cases} \gamma \text{ when } i = g \\ 0 \text{ otherwise} \end{cases} \\
\hat{X}H\hat{X}^T &= H
\end{aligned}$$

□

Theorem 1: *Given the Laplacian, Q, a diagonal matrix H whose entries equal γ, and the scaled eigenvector matrix $V = MX\sqrt{(H - \Lambda)}$, then:*

$$VV^T = MH - Q$$

PROOF:

Recall that M, is the diagonal matrix of vertex sizes, and $S^T S = M$, and $\hat{Q} = S^{-1} Q S^{-1}$. Multiplying out VV^T and simplifying, we find that:

$$\begin{aligned}
VV^T &= (MX\sqrt{(H-\Lambda)})(MX\sqrt{(H-\Lambda)})^T \\
&= MX(H-\Lambda)(MX)^T \\
&= MXHX^TM - MX\Lambda X^TM \\
&= S(SXHX^TS)S - S(SX\Lambda X^TS)S \\
&= S(\hat{X}H\hat{X}^T)S - S(SX\Lambda X^TS)S \\
&= SHS - S(\hat{X}\Lambda\hat{X}^T)S \\
&= MH - S\hat{Q}S \\
&= MH - Q
\end{aligned}$$

□

This equation leads to some very interesting corollaries. We can now relate the rows of V to the Laplacian Q, which is directly constructed from the graph's adjacency and degree matrices. Let ν_i represent the i^{th} row of V, v_i be vertex i and $deg(v_i)$ be the degree of v_i.

Corollary 1:

$$||\nu_i||^2 = m_{ii}\gamma - deg(v_i)$$

Proof: Using Theorem 1, we note that the diagonal entries of VV^T equal $(MH - Q)_{ii}$.

$$||\nu_i||^2 = \nu_i \nu_i^T = (MH - Q)_{ii} = m_{ii}\gamma - deg(v_i)$$

Corollary 2: *For $i \neq j$,*

$$\nu_i \nu_j^T = a_{ij}$$

Proof: Using Theorem 1, we note that the off-diagonal entries of $VV^T = A$:

$$\nu_i \nu_j^T = a_{ij}$$

The goal maximum sum vector partitioning is to find a set of vectors, T_h such that the Euclidean norm of their sum is as large as possible. Let $T_h = \sum_{v_i \in P_h} \nu_i$ and $\bar{E}_h$ be the total weighted sum of edges contained within a partition.

Theorem 2: *Let $||T_h||^2 = ||\sum_{v_i \in P_h} \nu_i||^2$ be the vector sum cost of vector partition h, and E_h be the edges cut in partition h of a graph partition Maximum sum vector partitioning is equivalent to graph partitioning.* $\sum_{h=1}^{k} ||T_h||^2 = \sum_{i=1}^{n} m_{ii}\gamma - \sum_{h=1}^{k} E_h$

Proof:

$$\begin{aligned} ||T_h||^2 &= ||\sum_{v_i \in P_h} \nu_i||^2 \\ &= \sum_{v_i \in P_h} ||\nu_i||^2 + \sum_{v_i, \nu_j \in P_h, \nu_i \neq \nu_j} \nu_i \nu_j^T \\ &= \sum_{v_i \in P_h} m_{ii}\gamma - deg(v_i) + \sum_{v_i, \nu_j \in P_h, v_i \neq v_j} a_{ij} \\ &= \sum_{v_i \in P_h} m_{ii}\gamma - E_h \end{aligned}$$

□

Summing up costs in all the partitions,

$$\begin{aligned} \sum_{h=1}^{k} ||T_h||^2 &= \sum_{h=1}^{k} \sum_{v_i \in P_h} m_{ii}\gamma - E_h \\ &= \sum_{i=1}^{n} m_{ii}\gamma - \sum_{h=1}^{k} E_h \end{aligned}$$

□

Here are a few other interesting results. We note that it is easy to derive the relationship between the angles between two vertices in terms of their graph properties. Likewise, the distance between two vertices can also be calculated using graph properties.

Corollary 3:

$$cos(\nu_i, \nu_j) = \frac{a_{ij}}{\sqrt{((m_{ii}\gamma - deg(v_i))(m_{jj}\gamma - deg(v_j)))}}$$

Proof:

$$\begin{aligned} cos(\nu_i, \nu_j) &= \frac{v_i v_j^T}{||v_i|| \cdot ||v_j||} \\ &= \frac{a_{ij}}{\sqrt{((m_{ii}\gamma - deg(v_i))(m_{jj}\gamma - deg(v_j)))}} \end{aligned}$$

□

Corollary 4:

$$||\nu_i - \nu_j||^2 = m_{ii}\gamma - deg(v_i) + m_{jj}\gamma - deg(v_j) - 2a_{ij}$$

Proof:

$$\begin{aligned} ||\nu_i - \nu_j||^2 &= ||\nu_i|| + ||\nu_j|| - 2\nu_i\nu_j^T \\ &= m_{ii}\gamma - deg(v_i) + m_{jj}\gamma - deg(v_j) - 2a_{ij} \end{aligned}$$

□

Corollary 5:

$$||\nu_i + \nu_j||^2 = m_{ii}\gamma - deg(v_i) + m_{jj}\gamma - deg(v_j) + 2a_{ij}$$

Proof:

$$\begin{aligned} ||\nu_i + \nu_j||^2 &= ||\nu_i|| + ||\nu_j|| + 2\nu_i\nu_j^T \\ &= m_{ii}\gamma - deg(v_i) + m_{jj}\gamma - deg(v_j) + 2a_{ij} \end{aligned}$$

□

3.2 Using the Results

We showed that the n-dimensional maximum vector sum problem is equivalent to the graph partitioning problem. In practice, it is too expensive to compute a large number of eigenvectors of large benchmarks. Furthermore, it is difficult to make use of that multi-dimensional information effectively. Therefore, we choose to use fewer eigenvectors (the first three) which then gives us an approximation of the partitioning problem, but one which is much easier to work with. Since the first eigenvector contains no useful information, we can use the second and third eigenvectors to form a planar embedding. The importance of the planar embedding comes from the spatial proximity of vertices.

3.3 Embeddings

Figure 3.1 shows an example of the 2-dimensional embedding formed by the second and third eigenvectors of the Laplacian of a graph. We also show two partitions found by our partitioner to graphically illustrate the motivation behind our hybrid partitioner. Vertices that belong to the same partition tend to be spatially close together in the embedding.

Figure 3.2 shows an example of the 2-dimensional embedding formed by the second and third scaled generalized eigenvectors of the Laplacian a structured mesh. In this case, it is even more obvious that spatial proximity is useful in obtaining good partitions of this graph.

There are a number of different variations on spectral partitioning: those based on the standard eigenvectors (traditional spectral partitioning), generalized eigenvectors (spectral partitioning with vertex sizes), scaled eigenvectors (maximum vector sum partitioning), and generalized scaled eigenvectors (maximum vector sum partitioning with vertex sizes). Our work concentrates on using the maximum sum vector partitioning formulation, so we use embeddings based on the generalized scaled eigenvectors, $V_d = MX_d\sqrt{(H_d - \Lambda_d)}$, with $\gamma = |\lambda_2| + |\lambda_d|$. Appendix A gives experimental verification of our embedding choice.

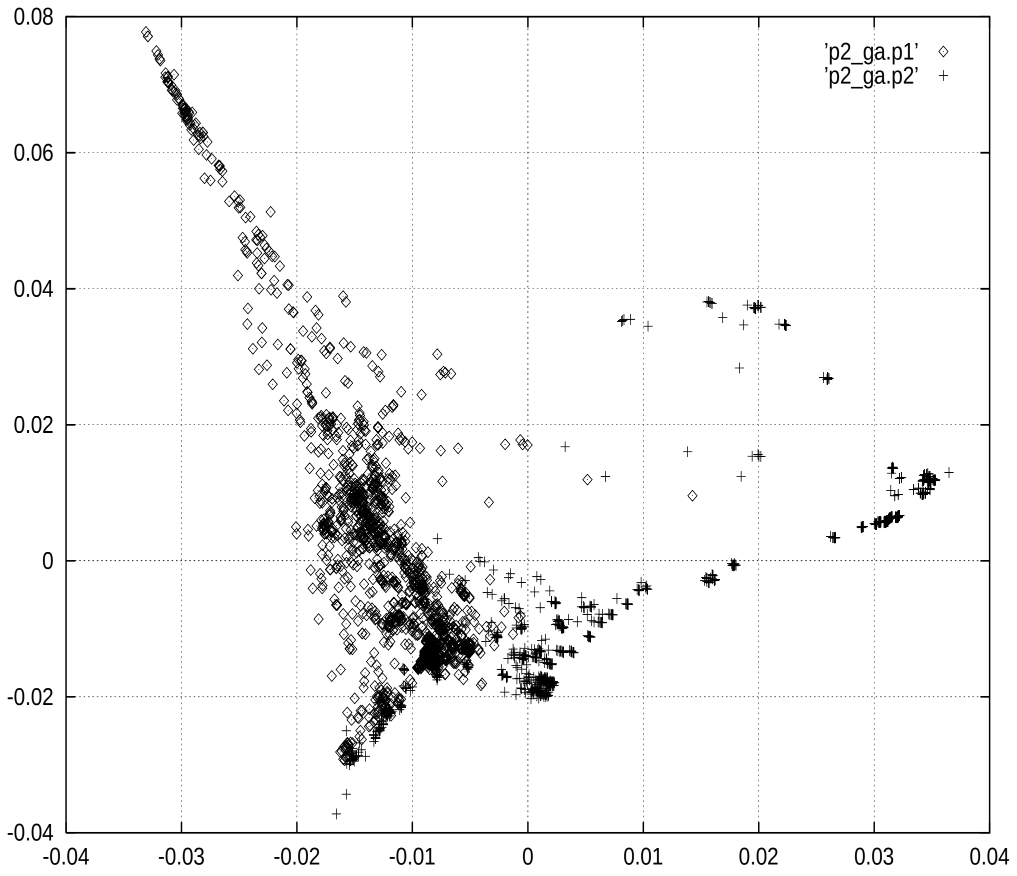

Figure 3.1: A 2-dimensional embedding of circuit p2_ga.

4 Hybrid Spectral/Iterative Partitioning

Researchers have tried to combine spectral and iterative partitioners by using the result of the EIG1 spectral partitioner [23] as the initial partition of an iterative improvement algorithm. The results were inferior to using random starts [25]. The primary problem is that the EIG1 algorithm only gives one starting point. Although it may be a very good one, it is not necessarily the best one for iterative improvement. We believe that a more integrated hybrid approach can utilize the strengths of both methods. In [11], they introduced a single-pass heuristic which constructed partitions based on the weighted sum of the edges cut and orthogonality (which was calculated using the eigenvectors of the graph). The algorithm was only used to construct partitions, and could not be used for iterative improvement. This paper significantly improves upon those previous attempts at combining spectral and iterative algorithms by directly using spectral information in a multi-level, k-way, KLFM iterative improvement algorithm.

The primary advantage of spectral algorithms is that they are able to find a globally optimal solution to a relaxed version of the partitioning problem. They have been found to perform well on partitioning problems using the ratio-cut cost function. However, spectral partitioning methods suffer from many glaring weaknesses which prevent them from performing well in constrained partitioning problems. These problems include:

- Constraints such as partition size, partition topology, and pin limits are difficult to incorpo-

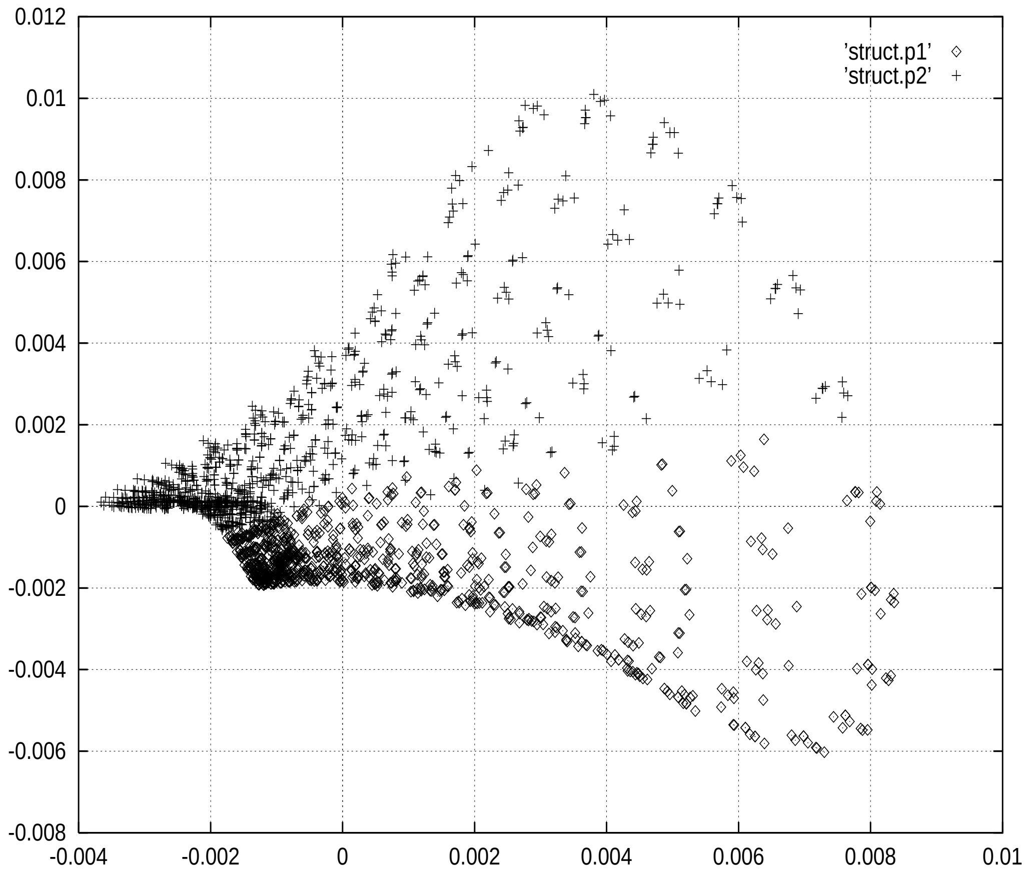

Figure 3.2: A 2-dimensional embedding of 'struct', a structured mesh.

rate.

- Converting a hypergraph into a graph will often result in a loss of information because the representation has changed. Optimal solutions for the resulting graph do not necessarily correspond to optimal solutions for the original hypergraph, although they may be very close.

- Only one deterministic solution is found. If that is a good solution, then there is no problem, however, if the solution is poor, then you are stuck with it.

- Arbitrary cost functions are either difficult or impossible to incorporate.

- The relaxed continuous-space solution may not be close to the optimal discrete partitioning solution.

With all of the above difficulties, it is no wonder that for bipartitioning with balance constraints, spectral algorithms have been found to be 4% worse than the standard KLFM algorithm and 18.8% worse than more advanced iterative partitioning algorithms [16]. What if we could combine spectral information with iterative improvement algorithms to obtain one unified method which combines the advantages of both methods? We could utilize the global information from a spectral algorithm within an iterative partitioning framework and gain the benefits of both methods.

5 Multi-way Partitioning

Most of the research done on iterative improvement has focused on 2-way partitioning. For k-way partitioners where $k > 2$, things are more complicated. We can decompose partitioning into two important phases: the generation of k initial partitions and the refinement of the solution. For iterative improvement partitioners, one obvious choice is to start with many random k-way partitions and improve upon the result. Another method makes use of an existing 2-way partitioner [29, 32]: Repeatedly apply a 2-way partitioning algorithm until the desired number of partitions is reached or constraints have been satisfied. One could iteratively create feasible partitions by breaking off pieces that satisfy constraints and then repeating the process on the remainder of the problem, or, recursively create balanced bipartitions until all partitions satisfy the constraints. Both methods have the consequence that the early partitionings may adversely affect the quality of later partitions.

Once k partitions have been created, iterative improvement can be used to refine the solution. Kernighan and Lin [29] suggested performing 2-way iterative improvement to each of the $\frac{k(k-1)}{2}$ pairs of partitions (we call this Pairwise KLFM).

Another approach, which we call Extended KLFM, involves extending the gain cost function in KLFM algorithms to directly support iterative moves from any one partition to any other partition [37, 26]. There are $k(k-1)$ possible directions of movement for vertices at each step.

We propose a new k-way iterative improvement method, Rotary KLFM which lies somewhere between the two improvement methods described above. In Rotary KLFM, each partition is used in turn as the target partition. The only moves considered are those in which vertices move into or out of the target partition, so there are $2(k-1)$ possible directions of movement at each step.

Currently, there are no direct comparisons among Pairwise KLFM, Extended KLFM, and Rotary KLFM. We compare our Rotary KLFM method to the Extended KLFM implementation of [33] in Section 7, however, because of underlying differences between the iterative improvement algorithms, the comparison is not equal. Each method has its own advantages. The advantage of Pairwise KLFM is that any existing 2-way iterative improvement heuristic can be used to improve k-way partitions. However, for each round of improvement, there are $\frac{k(k-1)}{2}$ iterative improvement passes, which could become prohibitively time-consuming as k increases. Rotary KLFM only needs k improvement passes for each round, while in Extended KLFM, there is only one pass for each round of improvement. In Pairwise KLFM, at each move step, there are only two possible directions for vertices to move. Rotary KLFM and Extended KLFM are quite similar, but Rotary KLFM is simpler because there are fewer possible move choices at each step and there are fewer gain updates to be made because there are $2(k-1)$ directions of movement compared to $k(k-1)$ in Extended KLFM.

5.1 Overview

Before we describe the details of our hybrid algorithm, we first present our multi-level partitioning framework, and describe its most important components. Figure 5.1 illustrates our multi-level partitioner. Spectral information can be used in the contraction algorithms, initial partitioning, and in the iterative improvement process. The eigenvectors and eigenvalues of a graph are computed during the initial partitioning. We can optionally recompute the eigenvectors in the Expand phase if desired. In our benchmarks, we recompute eigenvectors during expansion, unless otherwise noted.

```
ALGORITHM: MLH
INPUT: hypergraph, target number of partitions,$k$, maximum contracted size
OUTPUT: $k$-way partitioning solution
METHOD:
Read hypergraph
While (graph > maximum contracted size) {
    Contract graph
}
for i=1 to max_iterations {
    Initial $k$-way Partitioning
    Iterative Improvement
    While (graph not fully expanded) {
        Expand graph, map solution
        Iterative Improvement
    }
}
```

Figure 5.1: The multi-level partitioning algorithm.

Our Rotary KLFM iterative improvement algorithm (Figure 5.2) improves k-way partitions by iteratively working on one partition at a time. Each partition is used in turn as the target partition. The only moves considered are those in which vertices move from one of the $k-1$ other partitions into the target partition, or out of the target partition and into one of the $k-1$ other partitions. We will hereafter refer to this k-way KLFM algorithm as Rotary KLFM. We use spectral information in the updating of the neighbor gains to break ties and influence the move sequence. Figure 5.3 shows how moves are selected. Our implementation uses heaps to store vertex gains rather than buckets. This design decision was made in order to support arbitrary (non-integer) gains which were necessary in our hybrid spectral/iterative algorithm.

5.2 Using Spectral Information

There are three key areas in iterative improvement algorithms that can be augmented using spectral information:

1. Initial partition generation
2. Breaking ties when choosing the next move to make
3. Jumping out of local minima

Initial Partition

The partitioning problem requires us to divide our set of vertices into several distinct subsets. What we need to do is to use the spectral information in such a way that we can quickly and simply generate an initial partitioning.

```
ALGORITHM: Rotary KLFM, $k$-way Iterative Improvement
INPUT: hypergraph, initial $k$-way partitioning solution
OUTPUT: improved $k$-way partitioning solution
METHOD:
Do {
    For each partition $h$ {
        Initialize vertex gains for moves into and out of $h$
        While (there are unlocked vertices) {
            Select best gain move into or out of $h$
            Move vertex and lock in place
            Update gains of vertex neighbors
            If best cost, save this position
        }
        Rewind to best position
    }
} While (cost improved)
```

Figure 5.2: The Rotary KLFM improvement algorithm.

```
ALGORITHM: FM_Select - Select a vertex to be moved
INPUT: Hypergraph; target partition, $dest$; number of partitions, $k$
LOCALS: Into and Outof, arrays of $k$ heaps which contain the gains of
    vertices that can be moved into or out of the destination partition
OUTPUT: Best-gain move which satisfies balance constraints, or,
    first possible move if no moves can satisfy balance
METHOD:
best_gain = -INFINITY
for $h$=1 to $k$ {
    /* check vertices that can move from target into $h$ */
    gain = Into[$h$].Max()
    Store this move and set best_gain=gain if ((gain > best_gain)
        and (move from target to $h$ satisfies balance constraints))
    /* check vertices that can move out of $h$ and into target */
    gain = Outof[$h$].Max()
    Store this move and set best_gain=gain if ((gain > best_gain)
        and (move from $h$ to target satisfies balance constraints))
}
Return best move and gain satisfying balance, or the first move seen if
nothing satisfies balance
```

Figure 5.3: Move selection in our iterative improvement algorithm

Past work has focused on using the second eigenvector for generating two-way partitions [36, 22]. For example, in the EIG1 algorithm [23], each possible partitioning of vertices on the line was checked to find the best ratio-cut cost solution (Figure 5.4). Alpert and Yao [3] extended the concept to multiple dimensions and multiple eigenvectors by generating linear orderings using space-filling curves. Our approach lies somewhere between those two approaches.

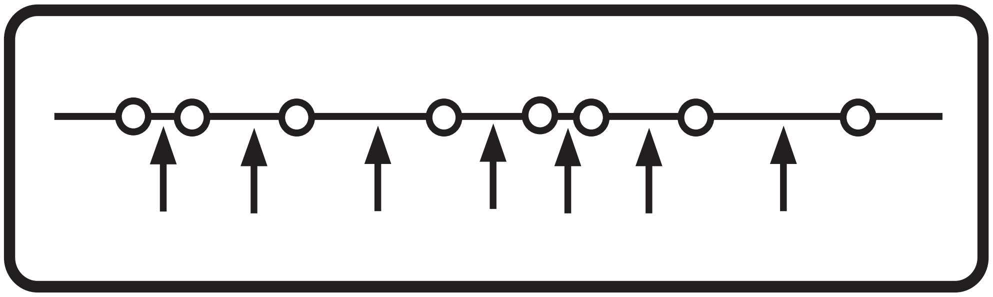
Figure 5.4: The EIG1 Spectral Ratio-Cut Partitioning Algorithm

Our focus is to solve the k-way partitioning problem for small values of k, such as 2,3, or 4. We use the second and third scaled eigenvectors to obtain a two-dimensional embedding (since the first eigenvector contains no useful information). Henceforth, it should be implicitly assumed that when we use the word eigenvectors we mean the scaled eigenvectors, V_d. We limit ourselves to planar embeddings for a number of practical as opposed to theoretical reasons:

- Planar embeddings are easy to subdivide.
- It is easy to generate orderings of vertices.
- Computation of the first three eigenvectors is fast.
- We can visually examine and understand our results.

More general methods for searching for solutions in a multi-dimensional solution space using vector probes were described in [18]. Since our vertices are embedded in a plane, we can systematically generate many different bipartitions using that placement. Using a planar embedding is a natural extension of [22], which used the second eigenvector to create an optimal placement of vertices on a line. The reason why we generate many initial partitions, rather than picking only the best one for iterative improvement is because the iterative improvement itself is very sensitive to the starting solution. Since our spectral partitioning result is so far removed from the original problem (due to conversion of a hypergraph to a graph, solving a relaxed problem instead of the ideal integer programming problem, and multiple levels of contraction), there is only a loose correspondence between initial and final solution costs. Figure 5.5 shows the initial and final solution costs for twenty iterations of our SWEEPB algorithm (which will be described later). We can readily see that the correspondence between initial and final solution costs is loose at best.

One of the first methods that comes to mind for generating initial partitions would be to divide the plane with a cutting line around the origin, which we call the SWEEPA algorithm (see Figure 5.6). In SWEEPA, we bisect the plane by stepping through a series of evenly-spaced angles which form a cutting line around the origin. Vertices that lie on one side of the line are placed in one partition, and the remaining vertices are placed in the other partition. We use every sweep step as an initial partition, rather than just picking the best one. This initial partition is then improved upon by iterative refinement, which will also enforce partition balance constraints.

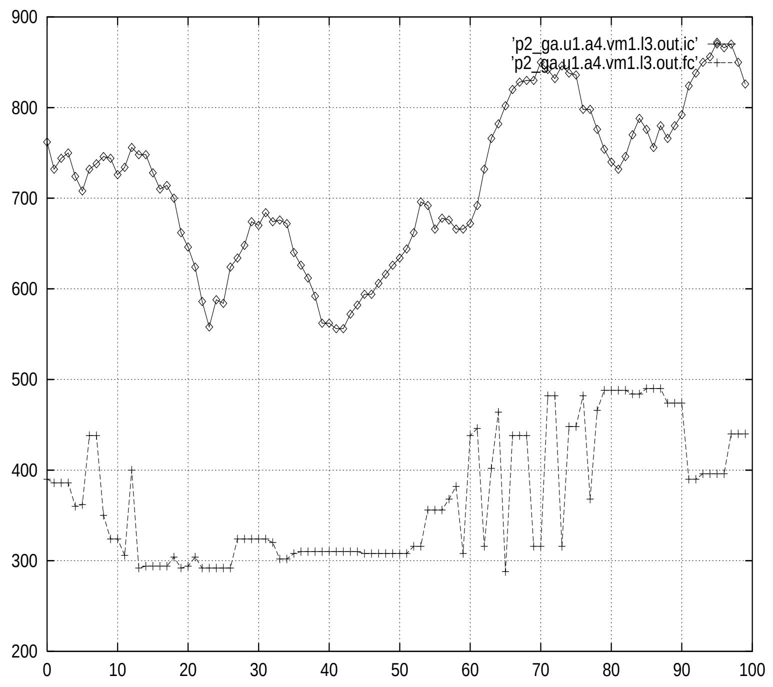

Figure 5.5: Initial and final solution costs for graph p2_ga, 3 levels of contraction

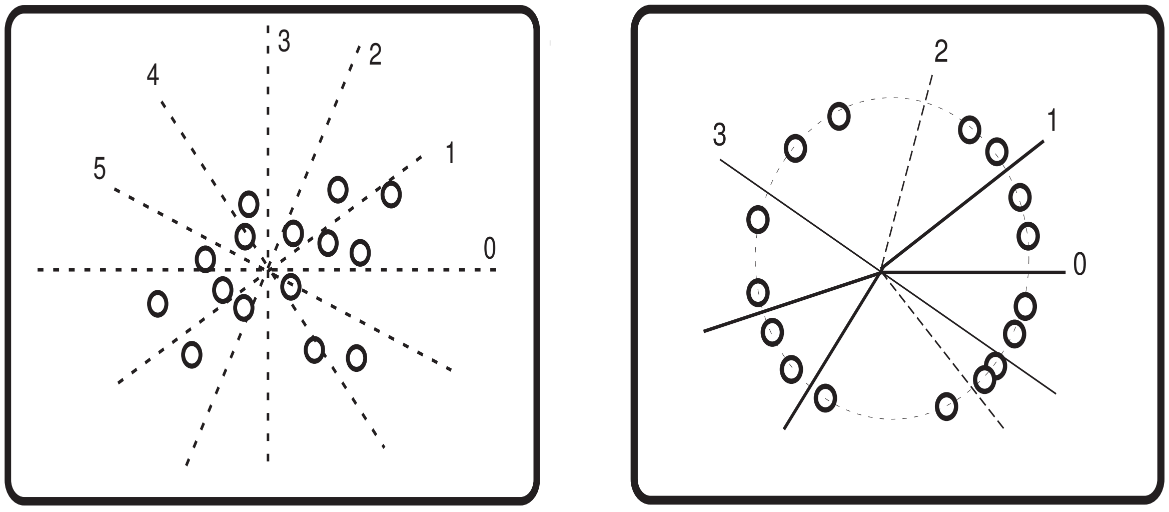

Figure 5.6: The SWEEPA algorithm on the left, and the SWEEPB algorithm on the right.

The SWEEPA algorithm is not as effective at generating a variety of initial partitions because often it generates the same initial partitioning for many steps if the planar embedding has large areas of empty space. In addition, balance constraints may initially be unsatisfied.

To address the problems of SWEEPA, we invented the SWEEPB algorithm. In the SWEEPB algorithm, we create a circular ordering of the vertices in the plane so that we can systematically create balanced initial partitions. The ordering of vertices is simply the angle of a vector with respect to the (1,0) vector, going in counter-clockwise direction. If we wish to create ρ initial partitions, we generate initial partitions by starting each partition $\frac{n}{k\rho}$ nodes apart. From the starting point, we place as many subsequent nodes as will fit into that partition, and when that partition is filled up, we continue travelling around the circular ordering, filling up each subsequent partition in order. For example, consider the SWEEPB diagram in Figure 5.6, where we would like four iterations: $\rho = 4$. If we want two balanced partitions, in the first iteration, P1 starts with the node number zero, whose angle is closest to the $(1, 0)$ vector. P1 is filled up with the first eight nodes, and $P2$ is filled up with the remaining nodes. In the second iteration, we start at vertex $\frac{16}{2 \cdot 4} = 2$ of the ordering. P1 is filled up with the first eight nodes from that starting point, and P2 contains the remaining nodes. On the third iteration, P1 starts at node 4, and on the last iteration, P1 starts at node 6.

Balance

We use soft constraints to enforce balance in partitions. The balance parameter is a user input, bal, which limits the minimum and maximum partition sizes as follows: $(1 - bal)||P_t|| \leq ||P_h|| \leq (1 + bal)||P_t||$. Here, P_h is an arbitrary partition, h, and $||P_t||$ is the target partition size, which is the total sum of the sizes of all vertices divided by k, the number of partitions desired. Unless otherwise noted, we always use $bal = 0.1$.

After every iterative improvement vertex move is made, we evaluate the cost of a solution. Typically, this cost is just the desired cost function, such as the ratio-cut cost, or the number of hyperedges cut. We add an auxiliary cost to that number to discourage imbalanced partitions. This auxiliary cost is equal to ten times the amount of imbalance of every partition.

Breaking Ties and Avoiding Local Minima

Iterative improvement methods are very sensitive to tie-breaking strategies when vertices are selected for movement. Rather than breaking ties arbitrarily, we break ties using spectral information. Consider what information we have. We have a two-dimensional placement of vertices in the plane. When we make iterative improvement moves, we can easily calculate distances using a wide variety of cost metrics. These distances could be either the distance from the vertex to be moved to the vector sum of the destination partitions, or it could be the distance from a vertex to its adjacent (connected one hop away in the graph, as opposed to spatially adjacent) neighbors. Using these ideas, we can come up with a number of tie-breaking strategies.

The total gain is a weighted combination of the standard cut gain and our spectral gain:

$$gain = \alpha \cdot cut\_gain + \beta \cdot spec\_gain \tag{5.1}$$

After each vertex move in iterative improvement, the neighbors of the moved vertex must have their gains updated. The cut_gain is the standard KLFM gain, which is the weight of the hyperedges that become uncut by a move.

Figure 5.7 shows the tie-breaking strategies we tried, and Table 5.1 shows our overall results for 2-way partitions on benchmarks p1_ga, p2_ga, t2, t3, t4, t5, and t6. In these experiments, we contracted to a maximum graph size of 200 nodes, using the $\alpha = 1.0$ and $beta = 0.1$.

The spectral gain cost we use is the magnitude of the vector sum, which we call the MVSA method. Let T_h be the sum of all the vectors (which represent vertices) in partition P_h. The spectral gain

```
ALGORITHM: Compute_Gain - Compute the gain of a move
INPUT: Hypergraph; vertex to update, $v$; last moved vertex, $lastv$
    last move destination, $lastp$; target partition, $dest$;
    number of partitions, $k$; gain method,$method$; weights $\alpha$ and $\beta$
OUTPUT: The gain of a vertex move
METHOD:
cut_gain = total weight of nets uncut when $v$ is moved to $dest$
if ($method$==COSAVG1)
    spec_gain = cosine of the angle from $v$ to the vector sum of all vertices in the destination
    if $v$ is in $dest$ then spec_gain=-1
if ($method$==COSN)
    spec_gain = 1+cosine of the angle from $v$ to $lastv$
if ($method$==COSAVG)
    spec_gain = 1+cosine of the angle from $v$ to the vector sum of all vertices in the destination
if ($method$==MVSAVG)
    spec_gain = magnitude of vector sum of $v$ to the vector sum of all vertices in the destination
if ($method$==MVSN)
    spec_gain = magnitude of vector sum of $v$ to $lastv$
if ($method$==DISTAVG)
    spec_gain = distance from $v$ to the vector sum of all vertices in the destination
if ($method$==DISTN)
    spec_gain = distance from $v$ to $lastv$
if ($v$ is in $lastp$) and (method != COSAVG1)
    then spec_gain=-spec_gain

gain = $\alpha\cdot$ cut_gain + $\beta\cdot$ spec_gain
Return gain
```

Figure 5.7: Calculating the hybrid spectral/iterative gain

Method	Actual Size	Unit Size
COSA	135.8	145.1
COSA1	138.0	142.1
COSN	133.4	145.6
MVSA	129.1	139.5
MVSN	131.4	147.9
DISTA	131.3	141.9
DISTN	134.8	144.6

Table 5.1: Comparison of various spectral-based cost metrics, using recalculated eigenvectors at each level. Results are the geometric mean of 2x the cut cost over seven benchmarks.

of a vector ν_i with respect to a destination partition P_h is $||\nu_i + T_h||^2$. The vector sums, T_h, of each of the k target partitions are calculated at the beginning of each pass, and are not dynamically updated as moves are made. The spectral gain is negated when the target destination is not the same partition as the last moved vertex. This encourages groups of vertices to move to the same target destination in connected groups, similar to the FM-CLIP algorithm [16].

When $\alpha = 1$, and $\beta = 0$, our algorithm behaves like the standard KLFM algorithm. When β is greater than 0, then the spectral gain information is used to influence the selection of the vertices. We experimented with two methods of incorporating spectral information. Our first method, HYBRID, is a simple tie-breaking strategy where we set $\alpha = 1.0$ and $\beta = 0.1$. The spectral information contributes only a small portion to the gain so that in effect, it is only used to break ties in gain.

Our second method, HYBRIDA, focuses on breaking out of local minima. The KLFM improvement algorithm makes greedy moves based on the cut gain. If we are able to make moves using some other objective cost function, then we can break out of local minima. In the HYBRIDA method, we alternately run our KLFM improvement algorithm using $\alpha = 1.0$, $\beta = 0.0$ and $\alpha = 0.0$, $\beta = 1.0$.

6 Issues in Multi-level Spectral Partitioning

There are a number of issues involved in multi-level spectral partitioning. Implementation choices at each step can have a significant effect on the final partitioning result. In this section, we consider the following implementation choices:

- Hypergraph to graph conversion - What is the best way to convert a hypergraph (circuit) into a graph so that it can be used in an eigensolver?
- Contraction algorithm - How should a (hyper)graph be contracted into a smaller one?
- Levels of hierarchy - What is the right number of levels to use to obtain good solutions?
- Disconnected graphs - What should be done about them?

6.1 Hypergraph to graph conversion

Figure 6.1a shows a hyperedge (net) from a hypergraph. In order to turn the hypergraph into a graph, all hyperedges must be transformed into edges (which connect exactly two vertices together). In the past, most researchers have simply performed a clique expansion [38] on hyperedges, shown in Figure 6.1b. In clique expansion, an edge is connected between all pairs of vertices incident on a degree d hyperedge, creating a total of $\frac{d(d-1)}{2}$ edges. With clique expansion, the primary factor affecting the outcome is the choice of edge weights. Some possibilities include $\frac{1}{d-1}$, ${\frac{2}{d}}^{1.5}$ [18] which is optimized for placement, or $\frac{4}{d(d-1)}$ [2] which is optimized for partitioning, or one which is optimized for k-way partitioning [21].

6.2 Star Graph

We develop a new hypergraph to graph conversion model, based on using the star graph (Figure 6.1c). The first use the star expansion in spectral partitioning was reported in [11]. Although the paper described star graphs, it did not present any significant quantitative or theoretical evaluation of the model. We derive an edge weighting function for the star graph, and we experimentally evaluate the performance of the star graph model. The star model creates a dummy vertex which is used to represent every edge in the hypergraph. Every actual vertex which is connected to the net is then connected to the dummy vertex representing the hyperedge. After sparse matrix computations have been performed, the dummy vertices are ignored. Since a star graph only creates d new edges for each hyperedge (where d is the degree of the hyperedge), matrices associated with star graphs are much sparser than those associated with clique expansions, which create $\frac{d(d-1)}{2}$ edges for each degree d hyperedge. On the other hand, this is offset by the fact that $|E|$ more rows are needed in the eigenvector solution.

Tables 6.1 and 6.2 show the time to compute the eigenvectors associated with the three smallest eigenvalues of various benchmarks. The time includes both the eigenvector computation and the time to translate the hypergraph into a graph. We see that the star graph performs consistently better when we use non-unit-size vertices. We also note that in the case where we perform three levels of contraction and compute the eigenvectors of the contracted graph, the execution times are about the same.

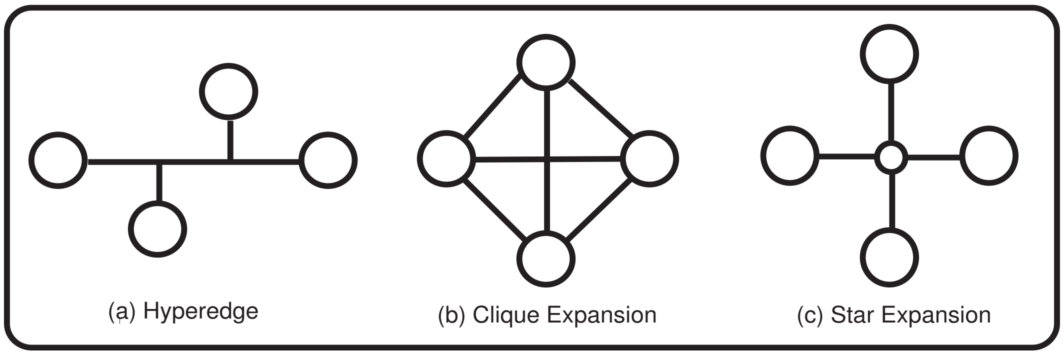

Figure 6.1: Approximating hypergraphs with graphs

We would like to carefully choose the weights of the star graph to optimize its performance for our problem.

A simple weighting model for star graphs is to pick the weight such that the sum of all the new edge weights is unit weight. Thus, $w = \frac{1}{d}$. We call this the StarD weighting model.

We could alternatively optimize our weights for the quadratic placement objective function. We derive the best weighting of edges in the star graph by examining the highest and lowest cost placements along a unit span, following the same method as [18]. The extreme cases are shown in Figure 6.2. The symbol 'X' is the dummy vertex which represents the hyperedge, and the circles represent single vertices. The concentric circle symbol represents multiple vertices placed at the same point. Let w represent the weight of a star graph's edge, which we will calculate below. There are two possible cases which we consider.

- **Unconstrained Solution** Figure 6.2a shows the minimum cost placement of one star net in the unit span. In this situation, all vertices are at the center except for two which are at the left and right extreme. The total quadratic placement cost is $2w(\frac{1}{2})^2 = \frac{w}{2}$. Figure 6.2b shows

	non-unit		unit	
Circuit	CliqueF	StarF	CliqueF	StarF
industry2	132.88	163.11	132.87	160.09
p1_ga	11.25	2.09	2.06	6.73
p2_ga	29.58	11.04	5.58	14.28
t2	60.09	4.75	11.42	15.40
t3	47.38	2.98	9.54	16.26
t4	41.45	4.13	14.61	15.46
t5	105.64	6.67	30.54	37.91
t6	43.06	6.18	9.54	12.99
GMean	46.98	7.40	12.77	20.34

Table 6.1: Time to compute first three eigenvectors of various benchmarks using no contraction.

the maximum cost placement on the unit span. Here, the total cost is wd. We can solve for the best value of w by solving $\frac{w}{2}wd = 1$, and we find that $w = \frac{\sqrt{2}}{\sqrt{d}}$.

- **Constrained Solution** In spectral partitioning, we would not expect Figure 6.2b to occur very often because we are optimizing the quadratic placement. Since a dummy vertex is only connected to vertices that are incident on that hyperedge, we would expect that in minimum-cost quadratic placements, the dummy vertex will lie between vertices. Figure 6.2c shows the maximum cost placement of vertices subject to the constraint that the dummy vertex lies between other vertices. The quadratic cost is $2w\frac{d}{2}(.5)^2 = \frac{wd}{4}$. The minimum-cost case is still the same, as shown in Figure 6.2a. We can solve for w using the equation $\frac{w}{2} \cdot \frac{wd}{4} = 1$. Our answer is $w = \frac{\sqrt{8}}{\sqrt{d}}$.

Both cases have a solution of the form $\frac{c}{\sqrt{d}}$. The constant c is unimportant because it is just a constant scaling factor. We call this model the StarF weighting model. In our experiments, we use $c = \sqrt{8}$.

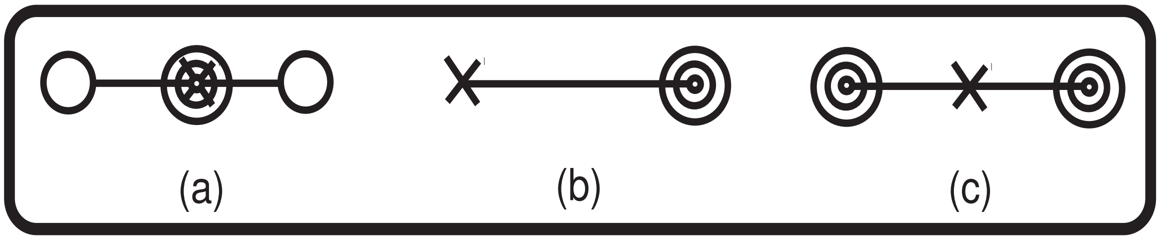

Figure 6.2: Star graph- extreme cost cases along the unit span

Table 6.3 is a quantitative comparison the star and clique graph models. We ran 20 iterations of our SWEEPB algorithm on the circuits p1_ga, p2_ga, t2, t3, t4, t5,t6, and industry2, contracting until circuits at most 400 nodes in size, and using the MVSA tie-breaking strategy. We sought to find balanced 2-way partitions with a 10% slack. Our results show that our StarF graph model produces excellent results, and is one of the top performers in the average and best cost tests for unit size vertices. This is the graph model we use in all of our benchmarks, unless otherwise specified. The CliqueD model is the best performer for tests using actual vertex sizes. Examining the best and worst case net models for the best solution cost using unit size vertices, there is only

	non-unit		unit	
Circuit	CliqueF	StarF	CliqueF	StarF
industry2	15.85	7.75	15.83	7.74
p1_ga	1.35	0.50	0.40	0.76
p2_ga	3.09	1.87	0.91	1.91
t2	3.84	0.72	1.09	0.92
t3	3.13	0.74	0.86	0.91
t4	2.38	0.76	1.10	1.02
t5	5.36	1.03	2.50	1.75
t6	3.93	0.78	1.22	1.23
GMean	3.76	1.11	1.44	1.46

Table 6.2: Time to compute first three eigenvectors of various benchmarks using three levels of contraction.

Graph Model	non-unit		unit	
	Average	Best	Average	Best
CliqueD	146.1	129.3	164.4	143.2
CliqueF	154.3	133.2	158.9	143.0
CliqueP	153.3	134.8	161.0	141.4
CliqueK	154.5	129.1	162.9	144.1
StarF	157.8	138.2	162.3	141.5
StarD	156.9	132.7	161.4	142.4

Table 6.3: Comparison of hypergraph to graph conversion techniques. Numbers are the geometric mean of the 2x the total cut cost over seven test circuits.

a 1.9% improvement. The graph model chosen does not appear to be a significant factor for these tests. Examining the best and worst case net models for the best solution cost using actual vertex sizes, we see an 7.8% span of improvement. The graph model appears to be more important when vertices are non-unit size. We note that in our benchmarks, the span of the actual vertex sizes vary significantly, having sizes that range over several orders of magnitude within the same circuit.

6.3 Contraction Method

In the past, researchers have used a variety of contraction methods to reduce the size of a problem. The contraction method plays a key role in not only reducing the problem size, but also influencing the final solution. We chose to use the heavy edge matching cost function [27]. This cost function contracts the edges with the highest weights, which would be the most undesirable to cut. We convert the hypergraph into a graph, sort the edges based on the edge weight, and match any unmatched pairs of vertices as edges are picked in order of decreasing cost.

6.4 Levels of Contraction

Graph contraction has a significant influence on solution quality. One of the contributing parameters of contraction is the clustering or matching method, which has been widely studied. Less attention

Max	average			best			-
Nodes	HYBRIDA	HYBRID	KFMC	HYBRIDA	HYBRID	KFMC	GMEAN
100	163.6	168.8	169.6	146.2	149.1	147.5	157.1
200	164.3	172.3	175.2	140.9	145.4	148.0	157.1
400	168.6	176.6	183.5	141.2	149.6	150.0	160.8
800	172.6	186.2	194.5	147.7	153.6	152.4	166.9
1600	177.7	192.8	217.3	147.8	162.1	162.3	175.2
unlimited	188.0	213.4	257.2	156.5	166.8	178.5	190.7

Table 6.4: Comparison of various levels of contraction using unit size vertices. Numbers are the geometric mean of 2x the total cut cost.

Max	average			best			-
Nodes	HYBRIDA	HYBRID	KFMC	HYBRIDA	HYBRID	KFMC	GMEAN
100	149.6	150.8	148.7	134.0	132.7	130.6	140.8
200	153.2	151.5	152.6	131.3	130.7	131.1	141.3
400	156.3	155.4	158.3	137.4	133.3	132.1	145.0
800	158.8	163.2	155.8	136.3	129.4	129.2	144.8
1600	179.1	180.5	170.2	138.0	137.1	132.0	154.8
unlimited	195.8	193.8	189.3	140.7	135.9	143.3	164.3

Table 6.5: Comparison of various levels of contraction, using actual vertex sizes. Numbers are geometric mean of 2x the total cut cost.

has been paid to the levels of contraction. This can have a dramatic effect on partitioning. As the number of levels of hierarchy increases, the initial problem solution becomes farther removed from the original problem. The best solutions may be precluded due to the contraction algorithm. This is offset by the fact that more levels of iterative improvement can be run when there are more hierarchical levels.

Many researchers have only used one level of contraction to perform partitioning [19, 13, 14], or a fixed number of levels, such as three [44]. Other researchers have chosen to contract the graph until the number of nodes in the graph reaches a target size, such as 35 nodes[5], 100 nodes [28] or 200 nodes [26]. For four-way partitioning, contraction was limited to 100 nodes in [5].

We ran some benchmarks using the graphs p1_ga, p2_ga, t2, t3, t4, t5, t6 for the unit-size and actual-size vertex tests, with the geometric mean of the total number of cuts shown shown in Table 6.4 and 6.5. In our tests, we evaluated various values of *Max Nodes*: 100, 200, 400, 800, 1600, and unlimited (which corresponds to no levels of contraction). We found that for two-way partitions, the best maximum number of nodes is in the 100 to 200 node range. We use a 200 node maximum contracted graph size for our 2-way 3-way and 4-way partitioning tests. Results are shown using our HYBRIDA, HYBRID, and KFMC, our implementation of the multi-level FM-CLIP algorithms.

6.5 Disconnected Graphs

Disconnected graphs cause problems for eigensolvers because solutions tend to lead to extremely unbalanced partitions (partitions where each connected component is in its own partition). We fix this problem by introducing dummy edges to connect the graph. These edges are introduced immediately before the eigensolver is invoked, and are then removed immediately after, so that they are never taken into account in any partition cut costs. For clique expansions, we pick an arbitrary vertex in each connected component and connect those chosen vertices together in a ring, in arbitrary order. For star graphs, we introduce a hyperedge which is connected to an arbitrary vertex in each of the components. Another approach to handling disconnected graphs is to use higher-order eigenvectors. Alpert [3] used up to ten eigenvectors in creating partitions. Shau [39] used different pairs of eigenvectors (1st and 2nd; 2nd and 3rd; 1st and 3rd) in his experiments. Pairwise selection of higher-order eigenvectors would be easy to implement in our partitioner, if necessary. Since we did not observe unusually poor performance for graphs using our simple connection method, we did not need to resort to using combinations of higher order eigenvectors.

7 Results on MCNC Benchmarks

We compared our hybrid algorithms against existing published results. The benchmark circuits we used are shown in Table 7.1. We show results for our HYBRID method, which uses the SWEEPB spectral initial partitioning algorithm and uses $\alpha = 1.0$ and $\beta = 0.1$ in the iterative improvement gain cost. We also show results for the HYBRIDA method, which also uses SWEEPB for initial partitioning, and alternates between $\alpha = 1.0$, $\beta = 0.0$ and $\alpha = 0.0$, $\beta = 1.0$ for each pass in order to jump out of local minima.

Circuit	Nodes	Nets
p1_ga	833	902
p1_sc	833	902
p2_ga	3014	3029
p2_sc	3014	3029
t2	1663	1720
t3	1607	1618
t4	1515	1658
t5	2595	2750
t6	1752	1541
biomed	6514	5742
s13207	8772	8651
s15850	10470	10383
industry2	12637	13419
industry3	15406	21923

Table 7.1: Benchmark Circuits

7.1 Implementation Details

Our partitioner was implemented in C++. We used the LASO code (written by David S. Scott) to compute eigenvectors and eigenvalues using sparse matrices. We replaced the traditional KLFM bucket implementation with heaps in order to support non-integer gains due to our hybrid cost function: $gain = \alpha \cdot cut\_gain + \beta \cdot spec\_gain$.

k	HYBRID	HYBRIDA	HYBRIDR	KFMC
2	41.9	40.2	42.4	38.7
3	88.7	92.2	89.5	94.9
4	122.9	124.3	132.8	132.6

Table 7.2: Hybrid algorithms have greater improvement as k increases. Numbers are the geometric mean cut cost over eight benchmarks with actual vertex sizes.

We contracted graphs until there were at most 200 nodes. We examined both the HYBRID tie-breaking algorithm as well as the HYBRIDA algorithm that alternates between spectral and iterative gain costs. The SWEEPB initial partitioning method was used with both HYBRID and HYBRIDA for 20 iterations ($\rho = 20$). We also report some results with the HYBRID tie-breaking strategy ($\alpha = 1.0$, $\beta = 0.1$) using random initial partitions instead of the SWEEPB method, which we call the HYBRIDR method.

7.2 Performance as k increases

Our partitioner performs well on 2-way partitioning, but is not better than the best 2-way partitioners. Appendix B gives the detailed benchmark results for the interested reader. The strength of our partitioner lies in combining global spectral information with iterative improvement. Global information becomes even more important in 3-way and 4-way partitioning, which is why our 3-way and 4-way results, are substantially better than other partitioners. Table 7.2 compares our hybrid algorithm with KFMC, our multi-level k-way implementation of the FM-CLIP method. We used the benchmarks p1_ga, p1_sc, p2_ga, p2_sc, t2, t3, t5, and t6, with actual vertex sizes, a balance factor of $bal = 0.1$, and the total hyperedges cut cost function. The HYBRID method is 7.6% *worse* than KFMC in 2-way partitioning. In 3-way partitioning, HYBRID is 7.0% better than KFMC, and HYBRID is 7.9% better than KFMC in 4-way partitioning.

Table 7.3 compares our hybrid algorithm with KFMC using unit size vertices. We used the benchmarks p1_ga, p2_ga, t2, t3, t4, t5, and t6, with a balance factor of $bal = 0.1$, and the total hyperedges cut cost function. For 2,3,4,5, and 6-way partitioning, HYBRID is better than KFMC by 6.2%, 9.0%, 27.3%, 31.3%, and 37.0% respectively.

As further evidence that spectral information is useful, especially as k increases, we can examine the results of the HYBRIDR method compared to the HYBRID method in Tables 7.2 and 7.3. The only difference between these two methods is the way initial partitions are generated. HYBRIDR uses random initial partitions, while HYBRID uses the SWEEPB method. We can see that for $k = 2$ and 3, HYBRIDR performs very well- sometimes slightly better, and sometimes slightly worse than HYBRID. However, as k increases, we see consistently better results with HYBRID, which uses the spectral embedding for generating initial partitions.

7.3 Comparison against GFM and Primal-Dual

We compared the total number of hyperedges cut by our hybrid algorithms to the GFM [33] and Primal-Dual (PD) algorithm [43]. These tests use the actual vertex sizes (which were obtained from Andrew Kahng of UCLA). We omitted graph t5 because in four-way partitioning, the largest

k	HYBRID	HYBRIDA	HYBRIDR	KFMC
2	69.7	70.9	69.7	74.0
3	103.2	104.0	101.8	112.5
4	123.4	123.7	130.2	157.1
5	146.8	143.5	158.8	192.7
6	166.0	165.6	175.0	227.4

Table 7.3: Hybrid algorithms have greater improvement as k increases. Numbers are the geometric mean cut cost over seven benchmarks with unit vertex sizes.

Benchmark	PD	GFM	KFMC	HYBRIDA	HYBRID
p1_ga	56	74	66	66	68
p1_sc	77	76	74	74	69
p2_ga	377	259	179	184	179
p2_sc	370	261	198	180	186
t2	81	81	81	81	81
t3	108	101	81	77	73
t5	80	85	88	89	83
t6	157	104	66	58	50
GMEAN	126.62	113.48	94.93	92.17	88.70

Table 7.4: Comparison of our HYBRID and HYBRIDA 3-way partitioning results against the GFM and PD partitioner using actual vertex sizes.

node in the circuit is larger than that allowed by balance constraints. Each partition was allowed a balance factor of $bal = 0.5$, and we used the CliqueD clique model.

Tables 7.4 and 7.5 show the results of our hybrid partitioner compared with the previously published results of [33]. The last row, GMEAN, is the geometric mean over all of the circuits. In the three-way partitioning, our HYBRIDA method gives a 23.1% improvement, and HYBRID gives a 27.9% improvement over GFM. In four-way partitioning, HYBRIDA is 48.7% better than GFM and HYBRID is 46.9% better than GFM.

As a control experiment, Tables 7.4 and 7.5 also show the results of 20 iterations of KFMC, our implementation of a multi-level, k-way FM-CLIP algorithm which uses our Rotary KLFM improvement method to determine whether the hybrid algorithm was really the source of improvement, rather than other factors, such as contraction method, levels of contraction, or Rotary KLFM. Our KFMC algorithm results are 11.6% better than GFM in 3-way partitioning, and 37.8% better than GFM in 4-way partitioning. Significant gains are made by our multi-level Rotary KLFM method. This leads us to conclude that a large portion of the substantial improvements in our k-way partitioning results come from the multi-level Rotary KLFM algorithm. Comparing our best hybrid algorithm results with KFMC, we observed a 7.0% improvement over KFMC in 3-way partitioning, and a 7.9% improvement over KFMC in 4-way partitionings.

Benchmark	PD	GFM	KFMC	HYBRIDA	HYBRID
p1_ga	102	107	98	87	91
p1_sc	107	110	90	86	99
p2_ga	459	335	229	215	228
p2_sc	426	354	234	236	226
t2	217	182	131	132	124
t3	170	162	104	101	105
t5	213	208	144	153	139
t6	189	145	103	67	68
GMEAN	205.41	182.70	132.59	122.85	124.34

Table 7.5: Comparison of our HYBRID and HYBRIDA 4-way partitioning results against the GFM and PD partitioner using actual vertex sizes.

Benchmark	HYBRID	HYBRIDA	KFMC	ML_F	GORDIAN
biomed	239	210	280	311	479
industry2	337	386	458	398	1179
industry3	763	752	784	830	1965
p1_ga	94	96	112	126	157
p2_ga	265	275	293	346	502
s13207	136	113	164	472	590
s15850	122	109	147	347	678
GMEAN	220.57	212.89	259.64	356.62	619.42

Table 7.6: Comparison of our HYBRID and HYBRIDA 4-way partitioning results against ML_F and GORDIAN. Results are 2x the total number of hyperedges cut using unit-size vertices.

7.4 Comparison against ML_F and GORDIAN

We compared our hybrid algorithm to the ML_F [5] and GORDIAN algorithms [30] as reported in [5]. We also report the results of our KFMC method for comparison. These tests use unit vertex sizes with a balance factor of $bal = 0.1$ and we used the StarF graph model.

Table 7.6 shows that our HYBRIDA partitioner is 22.0% better than KFMC, 67.5% better than the ML_F algorithm, and 191.0% better than GORDIAN in terms of the number of hyperedges edges cut. It is apparent that our Rotary FM algorithm provides significant improvements, and our hybrid spectral/iterative method is able to effectively use the global spectral information to substantially improve upon other partitioning methods.

8 Execution Time

The execution time of our partitioner is slower than that of other partitioners due to the use of heap data structures rather than buckets. Heaps were necessary in order to support non-integer gains in iterative improvement for our hybrid algorithm. Three-way partitioning execution time results (in seconds) are shown in Table 8.1. All our tests were run on a Sun Ultra 170, while GFM

Benchmark	HYBRIDA	HYBRID	GFM
p1_ga	84.9	55.1	30
p1_sc	78.9	54.4	36
p2_ga	747.0	351.8	222
p2_sc	489.6	354.9	211
t2	767.4	556.9	158
t3	499.5	372.3	92
t5	2346.0	1407.5	276
t6	1076.3	757.3	101

Table 8.1: Comparison of execution time in seconds against the GFM, 3-way partitions.

execution times are on a Sun Sparc 10.

9 Conclusion

We have developed a new hybrid spectral/iterative partitioning algorithm and have demonstrated that it performs better than the best known 3 and 4-way partitioners, using both unit vertex sizes and actual vertex sizes. The hybrid algorithm we have presented is only the first step in developing newer, more sophisticated iterative improvement algorithms. They key ideas that need further research include the use of new gain cost functions to influence move selection and new objective functions that allow iterative improvement algorithms to break out of local minima. Other areas of future research include new ways of using spectral information to generate initial partitions, using higher dimensions (more eigenvectors), generating orderings using other methods or cost functions, and investigating methods for speeding up hybrid partitioning.

References

[1] C. J. Alpert and A. B. Kahng. Geometric embeddings for faster (and better) multi-way netlist partitioning. Technical Report CSD TR-920052, UCLA, October 1992.

[2] C. J. Alpert and A. B. Kahng. Geometric embeddings for faster and better multi-way netlist partitioning. In *Proc. ACM/IEEE Design Automation Conference*, pages 743–748, 1993.

[3] C. J. Alpert and So-Zen Yao. Spectral partitioning: The more eigenvectors, the better. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, June 1995.

[4] Charles J. Alpert, Lars W. Hagen, and Andrew B. Kahng. A hybrid multilevel/genetic approach for circuit partitioning. In *Proceedings of the ACM/SIGDA Physical Design Workshop*, pages 100–105, 1996.

[5] Charles J. Alpert, Jen-Hsin Huang, and Andrew B. Kahng. Multilevel circuit partitioning. In *34th Design Automation Conference*, volume 34, 1997.

[6] E. R. Barnes. Partitioning the nodes of a graph. *Proceedings of Graph Theory with Applications to Algorithms and Computer Science*, pages 57–72, 1985.

[7] E.R. Barnes. An algorithm for partitioning the nodes of a graph. *SIAM Journal on Algorithm and Discrete Method*, 3:541–550, December 1982.

[8] E.R. Barnes, Anthony Vanelli, and James Q. Walker. A new heuristic for partitioning the nodes of a graph. *SIAM Journal on Algorithm and Discrete Method*, 1(3):299–305, August 1988.

[9] P. Bertin, D. Roncin, and J. Vuillemin. Programmable active memories: A performance assessment. In 1^{st} *International ACM/SIGDA Workshop on Field Programmable Gate Arrays, Berkeley, CA*, pages 57–59. ACM, February 1992.

[10] P. K. Chan, M. D. F. Schlag, and J. Y. Zien. Spectral k-way ratio-cut partitioning and clustering. *IEEE Trans. on CAD*, 13(9):1088–1096, September 1994.

[11] Pak K. Chan, M. D. F. Schlag, and Jason Y. Zien. Spectral-based multiway fpga partitioning. *IEEE Trans. on CAD*, 15(5):554–560, May 1996.

[12] Pak K. Chan, Martine Schlag, and Marcelo Martin. Borg: A reconfigurable prototyping board using field- programmable gate arrays. Technical Report UCSC-CRL-91-45, Board of Studies in Computer Engineering, University of California at Santa Cruz, Santa Cruz, CA 95064, November 1991.

[13] Chung-Kuan Cheng and Yen-Chuen Wei. An improved two-way partitioning algorithm with stable performance. *IEEE Transactions on Computer-Aided Design*, 10(12), December 1991.

[14] Jason Cong and M'Lissa Smith. A parallel bottom-up clustering algorithm with applications to circuit partitioning in vlsi design. In *Proc. ACM/IEEE Design Automation Conference*, pages 755–760, 1993.

[15] Alfred E. Dunlop and Brian W. Kernighan. A procedure for placement of standard-cell vlsi circuits. *IEEE Transactions on Computer-Aided Design*, 4(1):92–98, January 1985.

[16] Shantanu Dutt and Wenyong Deng. Vlsi circuit partitioning by cluster-removal using iterative improvement techniques. Technical Report CJA-PDW96, University of Minnesota, Dept. of Electrical Engineering, Minneapolis, MN, Nov. 1995.

[17] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, pages 175–181. ACM, 1982.

[18] Jonatahan Alexander Frankle. Circuit placement methods using multiple eigenvectors and linear probe techniques. Technical Report UCB/ERL M87/32, University of California, Berkeley, Electronics Research Laboratory, Berkeley, CA, May 1987. Ph.D. Dissertation.

[19] Jorn Garbers, Hans Jurgen Promel, and Angelika Steger. Finding clusters in vlsi circuits. In *ICCAD*, pages 520–523, 1990.

[20] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, NY, NY, 1979.

[21] Scott W. Hadley, Brian L. Mark, and Anthony Vannelli. An efficient eigenvector approach for finding netlist partitions. *IEEE Transactions on CAD*, 11(7):885–892, July 1992.

[22] Lars Hagen and Andrew Kahng. Fast spectral methods for ratio cut partitioning and clustering. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 10–12, 1991.

[23] Lars Hagen and Andrew Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on CAD*, 11(9):1074–1085, September 1992.

[24] Kenneth M. Hall. An r-dimensional quadratic placement algorithm. *Management Sciences*, 17(3):219–229, November 1970. The Institute of Management Sciences, Providence, RI.

[25] Scott Hauck and Gaetano Borriello. An evaluation of bipartitioning techniques. *submitted to, IEEE Transactions on CAD*, 1996.

[26] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301 * UC-405, Sandia National Laboratories, Albuquerque, New Mexico, 1993.

[27] George Karypis and Vipin Kumar. Analysis of multilevel graph partitioning. Technical Report 95-037, University of Minnesota, Dept. of Computer Science, August 1995.

[28] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report 95-035, University of Minnesota, Dept. of Computer Science, July 1995.

[29] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49(2):291–307, February 1970.

[30] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. Gordian: Vlsi placement by quadratic programming and slicing optimization. *IEEE Transactions on CAD*, 10(3):356–365, 1991.

[31] Balakrishnan Krishnamurthy. An improved min-cut algorithm for partitioning vlsi networks. *IEEE Transactions on Computers*, 33(5):438–446, May 1984.

[32] Roman Kuznar, Franc Brglez, and Krzysztof Kozminski. Partitioning digital circuits for implementation in multiple fpga ics. Technical Report Technical Report 93-03, MCNC Center for Microelectronic Systems Technologies, Research Triangle Park, NC, March 1993.

[33] Lung-Tien Liu, Mign-Ter Kuo, Shih-Chen Huang, and Chung-Kuan Cheng. A gradient method on the initial partition of fiduccia-mattheyses algorithm. In *ICCAD*, 1995.

[34] Daniel P. Lopresti. Rapid implementation of a genetic sequence comparator using field-programmable logic arrays. In *Proceedings of the Advanced Research in VLSI Conference at the University of Calfornia, Santa Cruz*, pages 138–152. The MIT Press, Cambridge, 1991.

[35] Tak-Kwong Ng, John Oldfield, and Vijay Pitchumani. Improvements of a mincut partition algorithm. In *ICCAD*, pages 470–473, 1987.

[36] Alex Pothen, Horst D. Simon, and Kang-Pu Lious. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal Matrix Analysis Appl.*, 11(3):430–452, July 1990.

[37] Laura A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–74, January 1989.

[38] D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. *Proceedings of the 9th Design Automation Workshop, Dallas, TX*, pages 57–62, June 1979.

[39] Jengie Shau. Spectral partitioning with diagonal perturbations. Technical report, University of California, Santa Cruz, Santa Cruz, CA, March 1997. M.S. Thesis.

[40] Douglas A. Thomae, Thomas A. Peterson, and David E. Van den Bout. The *anyboard* rapid prototyping environment. In *Proceedings of the Advanced Research in VLSI Conference at the University of Calfornia, Santa Cruz*, pages 356–370. The MIT Press, Cambridge, 1991.

[41] Yen-Chuen Wei and Chung-Kuan Cheng. A two-level two-way partitioning algorithm. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 516–519, 1990.

[42] Yen-Chuen Wei, Chung-Kuan Cheng, and Ze'ev Wurman. Multiple-level partitioning: An application to the very large-scale hardware simulator. *IEEE Journal of Solid-State Circuits*, 26(5):706–716, May 1991.

[43] C. W. Yeh, C. K. Cheng, and T. T. Lin. A general purpose multiple-way partitioning algorithm. *IEEE Transactions on CAD*, pages 1480–1488, 1994.

[44] Jason Y. Zien, Martine D. F. Schlag, and Pak K. Chan. Multi-level spectral hypergraph partitioning with arbitrary vertex sizes. In *ICCAD*, pages 201–204, 1996.

	Average Cost		Best Cost	
Method	Unit Size	Actual Size	Unit Size	Actual Size
Standard Eigenvectors	163.8	152.7	143.3	129.5
Scaled Standard	159.0	205.4	143.3	156.0
Generalized Eigenvectors	162.3	157.8	141.5	138.2
Scaled Generalized	158.2	150.7	139.5	129.1

Table A.1: Comparison of embedding types for balanced partition tests. Numbers are the geometric mean of 2x the hyperedges cut for seven benchmarks.

A Embedding Choices

We experimentally tested four spectral formulations with our HYBRID algorithm. We ran 20 iterations of HYBRID using the SWEEPB algorithm to generate initial 2-way partitions, with a maximum contracted graph size of 200 nodes and a balance factor of $bal = 0.1$. Table A.1 shows the geometric mean of the results from benchmarks p1_ga, p2_ga, t2, t3, t4, t5, and t6. The first row shows the results using the eigenvectors of the Laplacian matrix of a graph. The second row shows results using the scaled eigenvectors based on the maximum sum vector partitioning formulation [18, 3]. The third row shows the results using the eigenvectors of the generalized problem where vertex sizes are taken into account [44]. Finally, the fourth row shows the result for the generalized maximum sum vector partitioning. We can see that in terms of the best cost, in the tests where vertices are initially unit size, the generalized scaled eigenvector results are 2.7% better than the worst embedding choice. In tests using actual vertex sizes, the generalized scaled eigenvector formulation is 20.8% better than the worst choice.

B 2-Way Partitioning Comparison

We give our 2-way partitioning results in Table B.1. We compare our results against GFM [33], Strawman [25], and ML_C [5]. All partitioners allowed a balance factor of $bal = 0.1$, and used unit size vertices. Our results are on par with the best 2-way partitioners, but are not the best overall. The strength of our partitioner lies in combining global spectral information with iterative improvement. Global information becomes even more important in 3-way and 4-way partitioning, which is why our 3-way and 4-way results, are substantially better than other partitioners.

Benchmark	HYBRID(20)	HYBRIDA(20)	GFM(80)	Strawman(10)	ML_C(100)
balu	27	27	27	27	27
biomed	123	102	84	83	83
industry2	189	188	211	188	164
industry3	270	270	241	256	243
p1_ga	47	47	47	49	47
p2_ga	139	145	139	143	139
s13207	68	63	66	57	55
s1423	13	13	16	14	-
s1488	50	53	46	-	-
s15850	45	45	63	44	44
s35932	49	50	41	47	41
s38417	59	57	81	53	49
s38584	51	55	47	49	47
s9234	44	42	41	42	40
sioo	25	25	25	-	-
struct	47	34	41	33	-

Table B.1: Comparison of our HYBRID and HYBRIDA 2-way partitioning results against other 2-way partitioners. Results are the total number of hyperedges cut using unit-size vertices. The number in parenthesis next to the algorithm name denotes the number of iterative improvement runs.