Spectral Partitioning: The More Eigenvectors, The Better

Charles J. Alpert, Andrew B. Kahng and So-Zen Yao†

UCLA Computer Science Department, Los Angeles, CA 90095-1596
† Cadence Design Systems, San Jose, CA 94135

Abstract

The *graph partitioning* problem is to divide the vertices of a graph into disjoint clusters to minimize the total cost of the edges cut by the clusters. A *spectral* partitioning heuristic uses the graph's eigenvectors to construct a geometric representation of the graph (e.g., linear orderings) which are subsequently partitioned. Our main result shows that when all the eigenvectors are used, graph partitioning reduces to a new vector partitioning problem. This result implies that as many eigenvectors as are practically possible should be used to construct a solution. This philosophy is in contrast to that of the widely-used *spectral bipartitioning* (SB) heuristic (which uses a single eigenvector to construct a 2-way partitioning) and several previous multi-way partitioning heuristics [7] [10] [16] [26] [37] (which use k eigenvectors to construct a k-way partitioning). Our result motivates a simple ordering heuristic that is a multiple-eigenvector extension of SB. This heuristic not only significantly outperforms SB, but can also yield excellent multi-way VLSI circuit partitionings as compared to [1] [10]. Our experiments suggest that the vector partitioning perspective opens the door to new and effective heuristics.

1 Introduction

Automatic circuit partitioning has become an essential tool for such important VLSI CAD applications such as top-down hierarchical cell placement, simulation and emulation [4]. Systems with several million transistors entail problem complexities that are unmanageable for existing logic- and physical-level design tools. Thus, partitioning is used to divide the system into smaller, more manageable components, with the traditional goal being to minimize the number of signals which pass between the components.

A VLSI *circuit netlist* consists of a set of *cells* which are connected by *signal nets*. Each net consists of a unique *source* (an output terminal of a cell) and multiple *sinks* (input terminals of other cells, to which the signal must be propagated). A circuit netlist is typically represented by a *hypergraph* $H(V_H, E_H)$, where the set of vertices V_H corresponds to the cells, and the set of hyperedges E_H (i.e., subsets of V_H) corresponds to the signal nets. A cell $v \in V_H$ is incident to the net $e \in E_H$ if $v \in e$. Typical mathematical approaches to circuit partitioning (such as quadratic optimization or eigenvector computations) require the circuit to be represented as a graph $G(V, E)$, i.e., a hypergraph with $|e| = 2$ for each $e \in E$. We write each edge $e \in E$ as a pair of vertices. Many hypergraph-to-graph transformations have been proposed (see [4] for a detailed survey) including (i) creating a dummy vertex v' for each hyperedge $e \in E_H$, and inserting edges (v, v') in E for each $v \in v'$ [30]; (ii) mapping each hyperedge to a vertex in G and constructing an edge between two vertices if their corresponding hyperedges are incident to a common vertex [34]; and (iii) for each hyperedge e, constructing an edge (v, w) for every pair of vertices $v, w \in e$.

This last transformation is called the *clique* model; we (and most previous authors) adopt it since it is the only one which uses the same set of vertices in the graph and the hypergraph, so that a partitioning of the vertices of G directly corresponds to a partitioning of the circuit. The problem remains as to how to assign costs to edges in G. Ideally, the total cost of the cut edges should be one, corresponding to the cut of a single hyperedge, no matter how the vertices of that hyperedge's clique are partitioned. However, Ihler et al. [31] prove that such a "perfect" clique model is impossible, and Lengauer [32] shows that for any cost function there is some partitioning solution with $\Omega(\sqrt{|e|})$ deviation from the desired unit cost of cutting the hyperedge e. Hence, many different cost functions have been proposed for the clique model:

- The "standard" clique model [32] assigns cost $\frac{1}{|e|-1}$ to each clique edge; it is motivated by the linear placement into fixed locations at unit separation [11].
- The "partitioning-specific" model assigns cost $\frac{4}{|e|(|e|-1)} \cdot \frac{2^{|e|}-2}{2^{|e|}}$ to each clique edge so that the *expected* cost of each cut hyperedge is one.
- The "Frankle" model [19] proposed assigning cost $(\frac{2}{|e|})^{3/2}$ to each clique edge to address linear placement with a quadratic optimization objective. This model has been utilized in the partitioning heuristic of [10].
- Costs of $\frac{2}{|e|}$ [28], $(\frac{2}{|e|})^3$ [41], and $\frac{4}{|e|^2-|e|}$ [15] have also been proposed for the clique edges.

It remains open as to which model is best for a given application. For VLSI netlist partitioning, Alpert and Kahng [2] empirically showed that the Frankle and partitioning-specific models are comparable, and that both are better suited than the standard model for multi-way partitioning.

Given a circuit netlist, we assume that a transformation has been applied to generate the graph $G(V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$. The symmetric $n \times n$ *adjacency matrix* $A = (a_{ij})$ gives the costs of the edges in E, where $a_{ij} > 0$ is the cost of edge $(v_i, v_j) \in E$ and $a_{ij} = 0$ if $(v_i, v_j) \notin E$. Let $deg(v_i) = \sum_{j=1}^{n} a_{ij}$ be the *degree* of v_i. The $n \times n$ *degree matrix* $D = (d_{ij})$ is given by $d_{ii} = deg(v_i)$ and $d_{ij} = 0$ if $i \neq j$. The $n \times n$ *Laplacian* matrix of G is defined as $Q = D - A$.

Definition: A *k-way partitioning* of G is a set of non-empty *clusters* (subsets of V) $P^k = \{C_1, C_2, \ldots, C_k\}$ such that each $v_i \in V$ is a member of exactly one C_h, $1 \leq h \leq k$.

The most common graph partitioning objectives involve some combination of the number of *cut edges* and cluster *size balance*. We assume that cluster sizes must satisfy prescribed bounds, and that the objective is to minimize the total cost of all cut edges. Later, we discuss other possible partitioning objectives.

Min-Cut k-Way Graph Partitioning: Given the adjacency matrix A corresponding to a graph G, a prescribed number of clusters k, and lower and upper cluster size bounds L_h and W_h, find P^k that satisfies

$L_h \leq |C_h| \leq W_h$ for each $1 \leq h \leq k$ and minimizes

$$f(P^k) = \sum_{h=1}^{k} E_h \quad \text{where} \quad E_h = \sum_{v_i \in C_h} \sum_{v_j \notin C_h} a_{ij}. \tag{1}$$

In other words, E_h is the cut of cluster C_h. Notice that f counts the cost of each cut edge twice.

Min-cut graph partitioning is known to be NP-complete, and many heuristic methods have been proposed (see [4] for a survey). Spectral methods are well-established [7] [16] [18] [27] [36] and have also been the subject of extensive study in recent years [2] [5] [6] [10] [19] [23] [25] [29] [37] [40]. These methods use eigenvectors of the Laplacian or adjacency matrix to construct various types of geometric representations of the graph. We loosely categorize previous methods according to five basic representations:

- **Linear orderings (spectral bipartitioning):** The classic works of Hall [27] and Fiedler [18] gave theoretical motivation to the *spectral bipartitioning* (SB) heuristic, which is widely used in both the VLSI and scientific computing communities. SB takes the linear ordering induced by the second eigenvector of the Laplacian (also called the *Fiedler* vector) and splits it to yield a 2-way partitioning. Subsequent works have proposed variations of this algorithm [6] [36] [25] and [23] [40] analyzed its performance.

- **Multiple linear orderings:** Barnes [7] proposed a multiple-eigenvector extension to spectral bipartitioning. Barnes tries to map each cluster in a k-way partitioning to one of k eigenvectors while minimizing the rounding error according to a transportation formulation. If there is zero rounding error, the optimum cut value – equal to a weighted sum of the largest k eigenvalues (of the adjacency matrix) [16] – is obtained. An extension of Barnes' approach has been given by Rendl and Wolkowicz [37], and iterative improvement post-processing was proposed by Hadley et al. [26].

- **Points in d-dimensional space:** Hall [27] proposed using the coordinates of the second and third eigenvectors of the Laplacian to construct a 2-dimensional placement. Alpert and Kahng [1] extended this idea to higher dimensions, i.e., the i^{th} entries of d eigenvectors yield the coordinates of v_i in d-space. They used a spacefilling curve to induce a linear ordering of the embedded vertices, and then split the ordering into a multi-way partitioning via dynamic programming. Hendrickson and Leland [29] have also used this embedding; they use d eigenvectors to construct a partitioning with 2^d clusters. Arun and Rao [5] propose a bipartitioning heuristic that constructs a 2-dimensional embedding, then partitions to minimize the distance between the centroids of the two clusters.

- **Probe vectors:** Frankle and Karp [19] proposed an algorithm that searches for a good solution by defining *probe* vectors in the d-space spanned by the d best eigenvectors of Q. For each probe, they find the binary vector that maximally projects onto the probe in $O(n \log n)$ time. This vector corresponds to a bipartitioning solution. For the Max-Cut problem (in which the objective f is maximized), Goemans and Williamson [22] construct a set of d-dimensional vectors that correspond to the vertices. They choose a random probe vector and assign vectors with positive projection onto the probe into one cluster, and those with negative projection onto the probe into the other cluster. They prove a strong approximation bound on the expected performance of their heuristic.

- **Vectors in *d*-space:** Chan et al. [10] use the same embedded vertices as in [2] [27], but view each embedded vertex as a *vector* rather than as a point. Their KP algorithm constructs partitioning solutions using the directional cosine between two vectors as a similarity measure between vertices. Each cluster is constructed to minimize the directional cosine between vectors of the cluster and the "cluster center prototype" vector for that cluster. For the Max-Cut problem, Frieze and Jerrum [20] apply a simpler version of KP to vectors obtained from solving a relaxation of the Max-Cut objective.

In our work, we propose a new geometric representation that is based on a reduction from the graph partitioning problem to a *vector partitioning* problem. This representation is similar to that used by Chan et al. [10], but the coordinates are scaled by a function of the eigenvalue. The resulting vectors comprise a *vector partitioning* instance; the bijection between graph vertices and vectors permits us to develop a correspondence between graph and vector partitioning solutions. We make the following specific contributions:

- We propose a new vector partitioning formulation. A set of vectors is to be partitioned into k disjoint subsets, and the vectors in each subset are summed to form k *subset vectors*. The objective can be either to minimize or maximize the sum of the squared magnitudes of the subset vectors. Unlike the directional cosines-based approach of [10], our objective captures both vector magnitude and direction. We show that when all n eigenvectors are used, graph partitioning reduces to vector partitioning.

- This result motivates the use of as many eigenvectors as are practically possible. We propose a greedy ordering heuristic that extends SB to multiple eigenvectors and reduces to SB when only a single eigenvector is used. Our experimental results show that the information contained in additional eigenvectors of the graph's Laplacian yields significant improvements over both SB and its recursive implementation.

- We show that our heuristic also outperforms the multi-way partitioning heuristic of Chan et al. [10] which uses k eigenvectors to construct a k-way partitioning. Hence, we provide theoretical and empirical evidence that as many eigenvectors as practically possible should be used. Our experimental results suggest that more sophisticated vector partitioning heuristics hold much promise for graph and circuit partitioning applications.

The rest of our paper is as follows. Section 2 develops the relationship between eigenvectors and the min-cut objective. Section 3 presents the vector partitioning problem and proves the equivalence between graph and vector partitioning. We also extend this result to alternative graph partitioning objectives. Section 4 presents our ordering heuristic. Section 5 presents experimental results, and Section 6 concludes with directions for future research.

2 Eigenvectors and Min-Cuts

In this section, we review terminology and key results that have appeared in the literature.

Definition: For any k-way partitioning P^k, the corresponding $n \times k$ *assignment* matrix $X = (x_{ih})$ has $x_{ih} = 1$ if $v_i \in C_h$ and $x_{ih} = 0$ otherwise. Column h of X is the *indicator vector* corresponding to cluster C_h, and is denoted by $\vec{X}_h$.

In other words, X represents a k-way partitioning solution as a $0-1$ matrix with k columns and exactly n nonzero entries. Each row of X has sum one, and column h has sum $|C_h|$. A well-known result is that the min-cut objective f can be directly expressed in terms of the Laplacian and assignment matrices (see e.g., [10] [37]).

Theorem 1 $f(P^k) = trace(X^T Q X)$. □

The trace of a matrix is the sum of its diagonal entries, e.g., $trace(A) = \sum_{i=1}^{n} a_{ii}$. Theorem 1 directly follows from the result of Hall [27] that $E_h = \vec{X}_h^T Q \vec{X}_h$.

Definition: An n-vector $\vec{\mu}$ is an *eigenvector* of Q with *eigenvalue* λ if and only if $Q\vec{\mu} = \lambda\vec{\mu}$. We denote the set of eigenvectors of Q by $\vec{\mu}_1, \vec{\mu}_2, \ldots, \vec{\mu}_n$ with corresponding eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. The $n \times d$ *eigenvector matrix* $U_d = (\mu_{ij})$ has columns $\vec{\mu}_1, \vec{\mu}_2, \ldots, \vec{\mu}_d$ and the $d \times d$ *eigenvalue matrix* $\Delta = (\lambda_{ii})$ has diagonal entries $\lambda_{ii} = \lambda_i$ and 0 entries elsewhere. When $d = n$, we will use U for U_n and Δ for Δ_n.

We assume that the eigenvectors are normalized, i.e., for $1 \leq j \leq n$, $\vec{\mu}_j^T \vec{\mu}_j = ||\vec{\mu}_j||^2 = 1$. The eigenvectors of Q have many interesting properties, including the following [33][1]:

1. The eigenvectors are all mutually orthogonal, hence they form a basis in n-dimensional space.
2. Each eigenvalue λ_j of Q is real.
3. The smallest eigenvalue λ_1 is 0 and has a corresponding eigenvector $\vec{\mu}_1 = [\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}}]^T$.
4. If G is connected then $\lambda_2 > 0$. Furthermore, the multiplicity of 0 as an eigenvalue of Q is equal to the number of connected components of G.

For simplicity, we assume that G is connected. Because the eigenvectors form a basis, any n-vector $\vec{x}$ can be expressed in terms of this new basis. Since the eigenvectors are orthonormal, $U^T U = U U^T = I$, and we may write $\vec{x} = U U^T \vec{x} = U(U^T \vec{x})$, or equivalently $\vec{x} = \sum_{j=1}^{n} (\vec{\mu}_j^T \vec{x}) \vec{\mu}_j$.

Definition: The $n \times k$ *projection matrix* $\Gamma = (\alpha_{jh})$ is given by $\alpha_{jh} = \vec{\mu}_j^T \vec{X}_h$, and thus $\Gamma = U^T X$. Column h of Γ is given by $\vec{\Gamma}_h = U^T \vec{X}_h$. The magnitude of the projection of $\vec{X}_h$ onto $\vec{\mu}_j$ is given by α_{jh} .

[1] The first two properties also hold for the eigenvectors of the adjacency matrix A. Some early spectral partitioning works used the spectra of A (e.g., [7] [16]); however, the last two properties of the Laplacian make Q more convenient and hence preferable. All the results of this work can be equivalently established with A instead of Q (except Corollary 4 which follows from the third property). We believe that A and Q are theoretically equivalent, i.e., any spectral result can be derived from either the eigenvectors of A or Q, but this equivalence has not yet been established (or disproven).

Thus, $\vec{\Gamma}_h$ is the indicator vector for cluster C_h using the transformed eigenvector basis. Each indicator vector $\vec{X}_h$ can now be written as $U\vec{\Gamma}_h$, or equivalently, $\vec{X}_h = \sum_{j=1}^{n} \alpha_{jh}\vec{\mu}_j$. Notice that $|C_h| = ||\vec{X}_h||^2 = \sum_{j=1}^{n} \alpha_{jh}^2 = ||\vec{\Gamma}_h||^2$.

Finally, we can express the min-cut objective in terms of eigenvector projections.

Theorem 2 $trace(X^T QX) = trace(\Gamma^T \Delta\Gamma)$. $\square$

Theorem 2 follows from definitions: for each $\vec{\mu}_j$, $Q\vec{\mu}_j = \lambda_j \vec{\mu}_j$, hence $QU = U\Delta$ and

$$trace(X^T QX) = trace(X^T U\Delta U^T X) = trace((U^T X)^T \Delta(U^T X)) = trace(\Gamma^T \Delta\Gamma).$$

By combining Theorems 1 and 2 and performing the matrix multiplication, we observe:

Corollary 3 $f(P^k) = \sum_{h=1}^{k} \sum_{j=1}^{n} \alpha_{jh}^2 \lambda_j$. $\square$

This corollary is a multi-way extension of the bipartitioning result [19]: $E_1 = \vec{X}_1^T Q\vec{X}_1 = \sum_{j=1}^{n} \alpha_{j1}^2 \lambda_j$.

Using indicator vectors to represent partitioning solutions allows us to express the cost of a solution in terms of the projections of indicator vectors onto eigenvectors. We now define the *vector partitioning* problem, show how to create an instance of this problem, and prove that an optimal vector partitioning solution corresponds to an optimal min-cut graph partitioning solution.

3 The Vector Partitioning Problem

Definition: A k-way *vector partitioning* of a set of vectors Y is a set of k non-empty subsets $S^k = \{S_1, S_2, \ldots, S_k\}$ such that each $\vec{y} \in Y$ is a member of exactly one S_h, $1 \le h \le k$.

The vector partitioning objective can either be minimized or maximized. We first show how graph partitioning reduces to the min-sum vector partitioning problem, then show a reduction to max-sum vector partitioning. The latter formulation permits more intuitive heuristic algorithms.

Vector Partitioning: Given a set Y of n vectors, a prescribed number of subsets k, and lower and upper size bounds L_h and W_h, find S^k that satisfies $L_h \le |S_h| \le W_h$ for each $1 \le h \le k$ and optimizes

$$g(S^k) = \sum_{h=1}^{k} ||\vec{Y}_h||^2 \quad \text{where} \quad \vec{Y}_h = \sum_{\vec{y} \in S_h} \vec{y}. \tag{2}$$

We call $\vec{Y}_h$ the *subset vector* for S_h. Let $\vec{y}_i^d$ denote row i of $\Delta_d^{\frac{1}{2}} U_d$. Consider the d-dimensional vector partitioning instance with the vector set $Y = \{\vec{y}_1^d, \vec{y}_2^d, \ldots, \vec{y}_n^d\}$, in which each graph vertex v_i corresponds to a vector $\vec{y}_i^d$. (Observe that $\vec{y}_i^n$ is the indicator vector corresponding to $\{v_i\}$ in the scaled eigenspace.) We say that a graph partitioning $P^k = \{C_1, C_2, \ldots, C_k\}$ *corresponds to* a vector partitioning $S^k = \{S_1, S_2, \ldots, S_k\}$ if and only if $v_i \in C_h$ whenever $\vec{y}_i^d \in S_h$. We now show that when $d = n$ the min-cut graph partitioning and min-sum vector partitioning objectives are identical.

Theorem 4 Let $S = \{\vec{y}_1^n, \ldots, \vec{y}_n^n\}$, where $\vec{y}_i^n$ is row i of $\Delta^{\frac{1}{2}}U$, then if P^k corresponds to S^k, then $f(P^k) = g(S^k)$.

Proof: For a given $S_h \in S^k$, let $\vec{Y}_h^n = \sum_{\vec{y}_i^n \in S_h} \vec{y}_i^n$. We have

$$\begin{aligned} \|\vec{Y}_h^n\|^2 &= \|\sum_{\vec{y}_i^n \in S_h} \vec{y}_i^n\|^2 = \sum_{j=1}^{n}(\sum_{\vec{y}_i^n \in S_h} \sqrt{\lambda_j}\mu_{ij})^2 = \sum_{j=1}^{n}(\sqrt{\lambda_j}\sum_{v_i \in C_h} \mu_{ij})^2 \\ &= \sum_{j=1}^{n}(\sqrt{\lambda_j}(\vec{\mu}_j^T \vec{X}_h))^2 = \sum_{j=1}^{n}(\sqrt{\lambda_j}(\alpha_{jh}))^2 \end{aligned}$$

since membership of $\vec{y}_i^d \in S_h$ corresponds to membership of $v_i \in C_h$. Since $E_h = \sum_{j=1}^{n} \alpha_{jh}^2 \lambda_j$ (from Corollary 3), we have $\|Y_h^n\|^2 = E_h$. Hence $g(S^k) = \sum_{h=1}^{k} \|Y_h^n\|^2 = \sum_{h=1}^{k} E_h = f(P^k)$. □

Corollary 5 Min-cut graph partitioning reduces to min-sum vector partitioning, hence min-sum vector partitioning is NP-hard. □

It is not hard to show that min-sum vector partitioning is actually NP-complete, although optimum bipartitioning solutions have been found with time complexity polynomial in d [19] [5].

Corollary 6 $\|\vec{y}_i^n\|^2 = deg(v_i)$. □

This result directly follows from $\|\vec{Y}_h^n\|^2 = E_h$ Corollary 6 helps in understanding why the graph partitioning and vector partitioning formulations are equivalent. We see that the degree of each graph vertex can be computed from the magnitude of its corresponding vector, and further, when these vectors are added together, the magnitude of the resulting subset vector gives the cost of the edges cut by the cluster.

If we ignore the eigenvector $\vec{\mu}_1 = [\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}}]$, then Theorem 4 still holds, but our vector partitioning instance has a nice geometric property that may be useful in designing vector partitioning heuristics.

Corollary 7 Given a vector partitioning solution S^k for $Y = \{\vec{y}_1^d, \vec{y}_2^d, \ldots, \vec{y}_n^d\}$, if the first component of each $\vec{y}_i^d$ is discarded, then $\sum_{h=1}^{k} \sum_{\vec{y} \in S_h} \vec{y} = \vec{0}$. □

The result follows from the fact that each eigenvector $\vec{\mu}_2, \ldots, \vec{\mu}_n$ is orthogonal to $\vec{\mu}_1$, hence $\sum_{i=1}^{n} \mu_{ij} = 0$ for $2 \leq j \leq n$.

Good heuristics for vector partitioning with a *min-sum* objective must be somewhat unintuitive. When each subset should consist of a set of vectors that sums as closely as possible to the zero vector, the vectors in a given subset will point in many different directions and it is not clear whether two similar vectors should belong in the same subset. Contrast this to a *max-sum* objective where $g(S^k)$ is to be maximized and we seek subsets of vectors that point in the same general direction. This objective allows much more intuitive approaches to the problem. Equation (2) induces an obvious pairwise similarity measure for max-sum vector partitioning. If $\vec{y}_i$ and $\vec{y}_j$ sum to a vector of large magnitude (relative to the sizes of $\vec{y}_i$ and $\vec{y}_j$), then these vectors likely belong in the same subset. Graph partitioning has no analogous natural similarity measure between vertices (cf. the many ad hoc means in the literature, such as all-pairs shortest paths, $k-l$ connectivity [21], degree/separation [24], etc.).

To establish a reduction between min-cut graph partitioning and max-sum vector partitioning, we reformulate the min-cut objective as the maximization objective $nH - f(P^k)$ (following [19]), where H is some constant greater than or equal to λ_n (for now, the choice of H is inconsequential). Using Corollary 3, we formulate a new maximization objective for graph partitioning as

$$nH - f(P^k) = \sum_{h=1}^{k}(H|C_h| - \sum_{j=1}^{n} \alpha_{jh}^2 \lambda_j) = \sum_{h=1}^{k}(H \sum_{j=1}^{n} \alpha_{jh}^2 - \sum_{j=1}^{n} \alpha_{jh}^2 \lambda_j) = \sum_{h=1}^{k} \sum_{j=1}^{n} \alpha_{jh}^2 (H - \lambda_j). \quad (3)$$

The choice of $H \geq \lambda_n$ ensures that $nH - f(P^k) \geq 0$.

Definition: The $n \times d$ *scaled eigenvector matrix* $V_d = (\nu_{ij})$ is given by $\nu_{ij} = \mu_{ij}\sqrt{H - \lambda_j}$, i.e., by the matrix U_d with each column $\vec{\mu}_j$ scaled by $\sqrt{H - \lambda_j}$.

Theorem 8 Let $S = \{\vec{y}_1^n, \ldots, \vec{y}_n^n\}$, where $\vec{y}_i^n$ is row i of V_n. If P^k corresponds to S^k, then $nH - f(P^k) = g(S^k)$. □

Following the proof of Theorem 4, one can establish that $||\vec{Y}_h^n||^2 = \sum_{j=1}^{n} \alpha_{jh}^2 (H - \lambda_j)$. Combining this observation with Equations (3) and (2) yields $nH - f(P^k) = g(S^k)$, the desired result. Equivalent results established for min-sum vector partitioning also hold for the max-sum problem, i.e., Corollaries 5, 6, and 7.

An Example

Figure 1(a) gives a possible bipartitioning for a graph with five vertices. The eigenvalues of the Laplacian of this graph are

$$\lambda_1 = 0.000, \quad \lambda_2 = 2.298, \quad \lambda_3 = 2.469, \quad \lambda_4 = 8.702, \quad \lambda_5 = 10.531.$$

We choose $H = \lambda_5$ so that the last column of V_5 is zero, and obtain

$$U_5 = \begin{bmatrix} 0.447 & 0.309 & 0.530 & 0.452 & 0.467 \\ 0.447 & 0.131 & 0.468 & -0.532 & -0.530 \\ 0.447 & -0.880 & 0.000 & 0.159 & 0.000 \\ 0.447 & 0.131 & -0.468 & -0.532 & 0.530 \\ 0.447 & 0.309 & -0.530 & 0.452 & -0.467 \end{bmatrix} \quad V_5 = \begin{bmatrix} 1.451 & 0.886 & 1.505 & 0.612 & 0.000 \\ 1.451 & 0.377 & 1.329 & -0.719 & 0.000 \\ 1.451 & -2.526 & 0.000 & 0.215 & 0.000 \\ 1.451 & 0.377 & -1.329 & -0.719 & 0.000 \\ 1.451 & 0.886 & -1.505 & 0.612 & 0.000 \end{bmatrix}$$

The first five rows of V_5 yield our vector partitioning instance, $Y = \{\vec{y}_1, \ldots, \vec{y}_5\}$, but Corollary 7 permits us to discard the first column, and our choice of H permits us to discard the last column. Thus, the subset vectors corresponding to the bipartitioning $\{\{v_1, v_2, v_3\}, \{v_4, v_5\}\}$ are

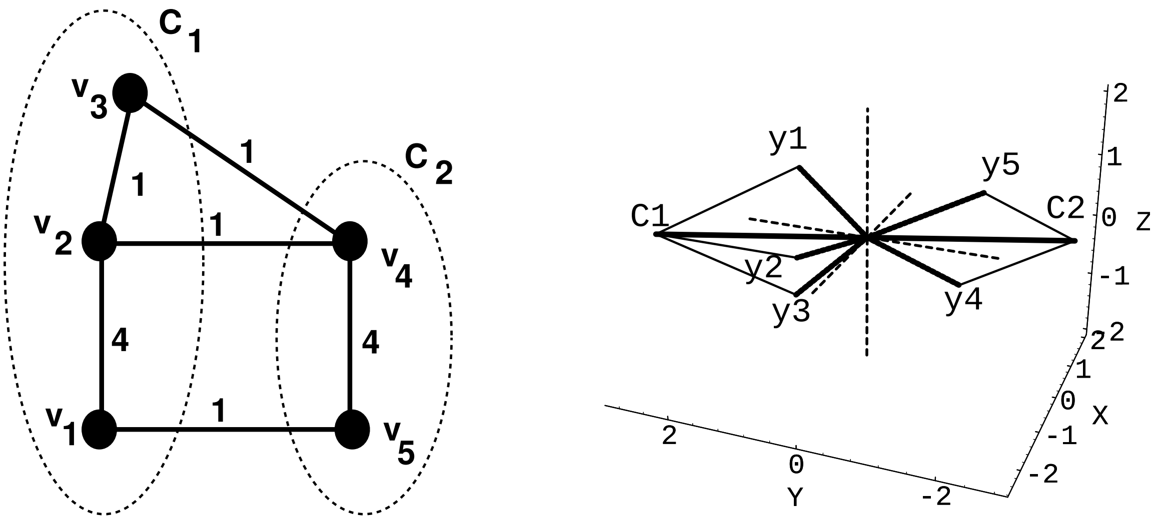
$$\vec{Y}_1 = \vec{y}_1 + \vec{y}_2 + \vec{y}_3 = [-1.263, 2.834, 0.108]^T, \quad \vec{Y}_2 = \vec{y}_4 + \vec{y}_5 = [1.263, -2.834, -0.108]^T.$$

The construction of these vectors is illustrated in Figure 1(b).

We can now verify Theorem 8 for this example. We have $||\vec{Y}_1||^2 = ||\vec{Y}_2||^2 = 9.637$. The sum of the magnitudes of the discarded components is (using $\alpha_{1h}^2 = \vec{\mu}_1^T \vec{X}_h = \frac{|C_h|^2}{n}$)

$$\sum_{h=1}^{k} \alpha_{1h}^2 (H - \lambda_1) = H(\alpha_{11}^2 + \alpha_{12}^2) = 10.531(\frac{9}{5} + \frac{4}{5}) = 27.378.$$

We have $g(S^2) = 9.637 + 9.637 + 27.378 = 46.655$. The graph partitioning maximization objective $nH - f(P^k)$ evaluates to $5 \cdot 10.531 - (3+3) = 46.652$, and we see that the objectives (except for rounding errors) are identical.



(a) (b)

Figure 1: (a) A bipartitioning solution for a graph with five vertices, and (b) the construction of the subset vectors $\vec{Y}_1$ and $\vec{Y}_2$ (shown as C1 and C2) for the corresponding vector partitioning instance.

Other Graph Partitioning Formulations

The standard minimum-cut objective is not the only graph partitioning objective that can be captured by our reduction to vector partitioning. We now observe how other graph partitioning objectives can be reduced to corresponding max-sum vector partitioning problems. In fact, any graph partitioning objective that is a function of the cluster cutsize (i.e., E_h) has an obvious corresponding vector partitioning objective.

Cluster Area Constraints: Instead of having cardinality bounds on cluster sizes, each vertex v_i can have an associated *weight* or area denoted by $w(v_i)$. The weight of a cluster C_h is defined as the sum of the weights of its modules, i.e., $w(C_h) = \sum_{v_i \in C_h} w(v_i)$, and the constraints of the graph partitioning formulation are $L_h \leq w(C_h) \leq W_h$. When the weight of vertex v_i is extended to be the weight of $\vec{y}_i^d$, the vector partitioning constraints are simply $L_h \leq w(S_h) \leq W_h$, for $1 \leq h \leq k$.

Minimum Scaled Cost: Chan et al. [10] proposed to minimize the *Scaled Cost* objective $f(P^k) = \frac{1}{n(k-1)} \sum_{h=1}^{k} \frac{E_h}{|C_h|}$ with no cluster size constraints; this objective reduces to the *ratio cut* objective $f(P^2) = \frac{E_1}{|C_1| \cdot |C_2|}$ for $k = 2$. For this objective, instead of using $nH - f(P^k)$ for the maximization objective, we use (after removing the constant $\frac{1}{n(k-1)}$ coefficient from $f(P^k)$)

$$kH - f(P^k) = \sum_{h=1}^{k} (H - \frac{E_h}{|C_h|}) = \sum_{h=1}^{k} (H - \frac{1}{|C_h|} \sum_{j=1}^{n} \alpha_{jh}^2 \lambda_j)$$

$$= \sum_{h=1}^{k} (\frac{H}{|C_h|} \sum_{j=1}^{n} \alpha_{jh}^2 - \frac{1}{|C_h|} \sum_{j=1}^{n} \alpha_{jh}^2 \lambda_j) = \sum_{h=1}^{k} \sum_{j=1}^{n} \alpha_{jh}^2 \frac{H - \lambda_j}{|C_h|}.$$

The proof of Theorem 4 established that $||\vec{Y}_h^n||^2 = E_h$; the analogous result for max-sum vector partitioning is $||\vec{Y}_h^n||^2 = H|C_h| - E_h$ which implies $\frac{||\vec{Y}_h^n||^2}{|C_h|} = H - \frac{E_h}{|C_h|}$. This is exactly the cost of cluster C_h in our new maximization objective $kH - f(P^k)$. The corresponding max-sum vector partitioning objective becomes $g(S^k) = \sum_{h=1}^{k} \frac{||\vec{Y}_h||^2}{|S_h|}$, which captures an "average vector" formulation: we wish to maximize the sum of the squared magnitudes of each subset vector divided by the number of vectors in the subset.

Minimum Largest Cut: Partitioning of logic onto multiple integrated-circuit devices (such as field-programmable gate arrays) might require minimizing the maximum degree of any cluster, rather than minimizing the sum of cluster degrees in the min-cut objective. The partitioning objective for this application is to minimize $f(P^k) = \max_{1 \le h \le k} E_h = \max_{1 \le h \le k} \alpha_{jh}^2 \lambda_j$, or equivalently, maximize $f'(P^k) = \min_{1 \le h \le k} \alpha_{jh}^2 (H - \lambda_j)$. The corresponding partitioning objective is to maximize $g(S^k) = \min_{1 \le h \le k} ||Y_h^n||^2$.

Max Cut: Another problem which has received much attention in the recent literature is the max-cut problem [13] [14] [35]:

$$\text{Maximize:} \quad f(P^k) = \sum_{h=1}^{k} |E(C_h)|.$$

The same mapping of $\vec{y}_i^n$ to row i of $\Delta^{\frac{1}{2}} U$ that was used to reduce min-cut graph partitioning to min-sum vector partitioning can be used to reduce the max-cut problem to max-sum vector partitioning.

4 Linear Ordering Algorithm

We now propose a simple, greedy graph partitioning heuristic that utilizes the corresponding max-sum vector partitioning instance. Instead of explicitly solving the vector partitioning problem, we construct a *linear ordering* of the vectors, which corresponds to a linear ordering of the graph vertices. Previous work [1] [25] [38] has shown the utility of ordering-based partitionings for VLSI applications. In addition, the idea of constructing an ordering and splitting it into a partitioning is at the heart of spectral bipartitioning (SB) [7] [36]. Our construction seeks to combine d distinct eigenvectors (where d is as large as practically possible) into a single vertex ordering that utilizes all of the eigenvector information. When $d = 1$, our approach reduces to spectral bipartitioning, i.e., the linear ordering is the same as the sorted Fiedler vector.

In a good vector partitioning solution S^k, any given subset of vectors S_h will consist of vectors that sum to a vector of large magnitude. Figure 2 describes our MELO (Multiple Eigenvector Linear Orderings) heuristic, which greedily seeks to construct such a subset. MELO first constructs the d-dimensional vector partitioning instance and initializes the set S to empty (Steps 1-2). MELO then adds to S the vector with largest magnitude, removes this vector from Y, and labels the vector's corresponding vertex first in the ordering. Each subsequent iteration through Steps 4-6 chooses the "best vector" $\vec{y}_i^d$ remaining in Y, i.e., the one that maximizes the magnitude of the sum of vectors in S plus $\vec{y}_i^d$. Thus, at any stage of the MELO execution, S should consist of a reasonably good subset for vector partitioning. MELO's time complexity is no worse than $O(dn^2)$, with some speedups possible; we discuss this in detail in the last section.

The MELO Algorithm
Input: $G(V,E) \equiv$ Undirected weighted graph $d \equiv$ Number of eigenvectors to compute **Output:** Linear ordering of V
1. Compute the d smallest eigenvectors of the Laplacian of G and construct the matrix V_d. 2. Set $Y = \{\vec{y}_1^d, \vec{y}_2^d, \ldots, \vec{y}_n^d\}$, where $\vec{y}_i^d$ is row i of V_d. Set $S = \emptyset$. 3. **for** $j = 1$ **to** n **do** 4. Find $\vec{y}_i^d \in Y$ such that the magnitude of the vector $\sum_{\vec{y} \in S \cup \{\vec{y}_i^d\}} \vec{y}$ is maximum. 5. Add $\vec{y}_i^d$ to S and remove $\vec{y}_i^d$ from Y. 6. Label v_i as vertex j in the ordering.

Figure 2: The MELO (Multiple Eigenvector Linear Orderings) Algorithm

The idea of iteratively constructing a cluster (i.e., subset of vectors) seems to be the most direct method for constructing a linear ordering. MELO may not seem much different from a graph traversal (cf. [3]), e.g., choose some vertex to be in its own cluster and iteratively add vertices to the cluster to minimize the cluster degree. However, with our approach each vector contains global partitioning information; the set of edges connected to any vertex in the original graph representation comprises strictly local information. Graph traversals often have many equivalent choices of vertices, while ties are extremely rare in MELO.

We may also motivate MELO's greedy approach in terms of the maximization objective $nH - f(P^k)$ used in the reduction to max-sum vector partitioning. For $k = 2$, the degree of the first cluster is the same as the degree of the second cluster (i.e., $E_1 = E_2$) and the maximization objective equivalently becomes $H - \frac{f(P^2)}{2} = \sum_{j=1}^{n} \alpha_{j1}^2 (H - \lambda_j)$. Since in practice we have only d of the eigenvectors,[2] we cannot evaluate all the terms in this summation; however, we can approximate it by using the first d terms (since these are the most dominant terms), i.e.,

$$H - \frac{f(P^2)}{2} \quad \propto \quad \sum_{j=1}^{d} \alpha_{j1}^2 (H - \lambda_j). \tag{4}$$

Assuming that this approximation is reasonable (which we observed in practice by the distribution of

[2] We use the Lanczos algorithm [39] to compute the Laplacian eigenvectors. When computing the eigenvectors with the smallest corresponding eigenvalues, $\vec{\mu}_i$ will always converge faster than $\vec{\mu}_j$ if $i < j$, e.g., it will never be the case that say the $\vec{\mu}_{20}$ is known but $\vec{\mu}_{10}$ is not. Thus, since we wish to utilize as much information as possible, we assume that if the eigenvector $\vec{\mu}_d$ is available, the eigenvectors $\vec{\mu}_1$ through $\vec{\mu}_{d-1}$ are as well.

eigenvalues), if we can find a cluster C_1 that maximizes this summation, then the partitioning objective $H - \frac{f(P^2)}{2}$ should be close to optimal. A reasonable algorithm for constructing C_1 is to begin with $C_1 = \emptyset$, and then iteratively add the vertex to C_1 that optimizes the approximation in Equation (4). Continuing until every vertex is a member of C_1 induces a linear ordering of vertices; this is exactly the MELO construction.

Observe that when $d = 2$, this algorithm is identical to spectral bipartitioning. By Corollary 7, we can ignore $\vec{\mu}_1$, so the right side of Equation (4) reduces to $(\vec{\mu}_2^T \vec{X}_2)^2(H - \lambda_2)$. MELO iteratively maximizes this sum by choosing the largest coordinate in $\vec{\mu}_2$, followed by the second largest, and so on. This corresponds to constructing a linear ordering by sorting the coordinates of $\vec{\mu}_2$. The success of SB shows that the approximation in Equation (4) is reasonable, even for $d = 2$. As d increases, the approximation becomes successively better, so MELO comes closer to optimizing the actual partitioning objective. Both SB and MELO construct a linear ordering based on this approximation, but MELO extends SB to multiple eigenvectors by combining multiple eigenvector orderings into a single linear ordering.

We have not yet discussed how to choose H. When $d = n$ the choice of H is inconsequential, but since $d \neq n$ in practice, the choice of H significantly affects the linear ordering. Indeed, the accuracy of the approximation in Equation (4) significantly depends on H. We tried the following different schemes for scaling the eigenvectors; experimental data for each scheme appears in the next section.

- **Scheme #1:** $H = \infty$. This is equivalent to no scaling, i.e., the eigenvector projections are equally weighted.
- **Scheme #2:** $H = \lambda_d + \lambda_2$. Here H is chosen to be relatively close to the largest used eigenvalue, ensuring that the $\sqrt{H - \lambda_j}$ scaling factors will vary considerably with j.
- **Scheme #3:** Instead of using MELO in the context of Equation (4), we attempt to preserve the original projection weights from Corollary 3, i.e., $f(P^2) = \sum_{j=1}^{n} \alpha_{j1}^2 \lambda_j$. Recall that $\sum_{j=1}^{n} \alpha_{j1}^2 = |C_1|$ is a constant, hence the coefficients for the smaller λ_j terms should be as large as possible. Instead of minimizing $\sum_{j=1}^{n} \alpha_{j1}^2 \lambda_j$, we maximize $\sum_{j=1}^{d} \frac{\alpha_{j1}^2}{\lambda_j}$. With this objective, each pair of α_{j1}^2 terms remains in the same proportion as in Corollary 3.
- **Scheme #4:** The effect that maximizing the Equation (4) summation has on minimizing cut cost depends on the accuracy of the approximation. To minimize the error of this approximation, we require that the sum of the "contributions" of the unused $n-d$ eigenvectors is zero, i.e., $\sum_{j=d+1}^{n} \alpha_{j1}^2 (H - \lambda_j) = 0$. This occurs when [19]
$$H = \frac{E_1 - \sum_{j=1}^{d} \alpha_{j1}^2 \lambda_j}{|C_h| - \sum_{j=1}^{d} \alpha_{j1}^2}.$$
Thus, after Step 6 in Figure 2, H is recomputed using C_1 as the set of vertices corresponding to vectors in S, and the coordinates of the vectors in Y and S are then readjusted. This scheme does not increase the time complexity of MELO; however, it may make possible speedups infeasible.

5 Experimental Results

In this section, we present four sets of experiments that we performed with MELO:

- Comparisons of the weighting schemes proposed in the previous section,
- Comparisons with different values of d,
- Multi-way partitioning comparisons with recursive spectral bipartitioning (RSB), KP [10], and SFC [1] algorithms,
- Balanced 2-way partitioning comparisons with SB [25] and PARABOLI [38].

Our experiments use the set of ACM/SIGDA benchmarks listed in Table 1 (available via the World Wide Web at http://ballade.cs.ucla.edu/~cheese). The eigenvector computations were performed using LASO2 code [39], with a driver provided by the authors of [10]. Before the eigenvectors were computed, each netlist was transformed into a graph using the partitioning-specific clique model. We also discuss use of the standard and Frankle clique models.

Test Case	# Modules	# Nets	# Pins
19ks	2844	3282	10547
bm1	882	903	2910
prim1	833	902	2908
prim2	3014	3029	11219
test02	1663	1720	6134
test03	1607	1618	5807
test04	1515	1658	5975
test05	2595	2750	10076
test06	1752	1541	6638
balu	801	735	2697
struct	1952	1920	5471
biomed	6514	5742	21040
s9234	5866	5844	14065
s13207	8772	8651	20606
s15850	10470	10383	24712
industry2	12637	13419	48404

Table 1: Benchmark circuit characteristics.

To generate multi-way partitionings from MELO orderings, we apply the "DP-RP" algorithm of [1]. DP-RP accepts a vertex ordering and returns a *restricted partitioning*, i.e., a k-way partitioning such that each cluster is a contiguous subset of the ordering. DP-RP uses dynamic programming to find the optimal set of $k-1$ splitting points in the ordering; it applies to all of the partitioning objectives that we have discussed. We choose to minimize the *Scaled Cost* objective since it has no size constraints, provides a single quantity that measures the quality of a linear ordering, and permits comparisons to previous algorithms. With the Scaled Cost objective and no cluster size bounds, DP-RP has $O(kn^2)$ time complexity.

Our first experiments compare MELO orderings using the four proposed weighting schemes. For each scheme, we generated ten linear orderings for each benchmark by using the d eigenvectors with smallest eigenvalue (ignoring $\vec{\mu}_1$) for $1 \leq d \leq 10$. DP-RP was applied to each of the ten orderings, and the best Scaled Cost values that were observed are reported in Table 2. The Scaled Cost values did not vary much for the different schemes, although the second and third schemes performed slightly better. The rightmost column of Table 2 reports the average percentage improvement of Scheme #2 versus the other schemes. Positive improvement is indicated by a +; if the improvement is negative, we report the percent improvement of the superior scheme versus Scheme #2 and indicate this with a −. The percentage improvements of Scheme #2 as a function of k are given in the last rows of the Table. From the bottom right of the Table, we see that #2 averages 1.70%, 0.01%, and 1.41% improvement over Schemes #1, #3, and #4 respectively. The results for the second and third schemes are indistinguishable; we choose the second scheme for our remaining experiments since it is derived directly from the vector partitioning instance. The fourth scheme performed surprisingly poorly; however, we observed that although H fluctuated, it remained considerably higher than $\lambda_2 + \lambda_d$, i.e., the eigenvectors were not scaled very significantly (we believe that this scheme would be more successful for larger d). These results indicate that (at least for the MELO ordering construction) the best weighting strategy is to make H small in order to maximize scaling.

Our second set of experiments explores the relationship between the quality of MELO partitionings and the choice of d. Table 3 gives Scaled Cost values for the single eigenvector ordering (1), i.e., the Fiedler vector, the MELO ordering that uses ten eigenvectors (10), and the best result obtained from the ten MELO orderings constructed with $1 \leq d \leq 10$ eigenvectors $(1 - 10)$ for Scheme #2. We observe that running MELO using 10 eigenvectors yields significantly better Scaled Costs than just using the Fiedler vector. Overall, the the $1 - 10$ orderings averaged 32.6% improvement over the Fiedler ordering but only 6.43% improvement over the $d = 10$ orderings. This illustrates that SB can be significantly improved by using more than one eigenvector to compute a linear ordering. In addition, the last two rows of Table 3 indicate that most improvement occurs for the smaller k values, suggesting that the 10 orderings are of high quality "from end to end". In other words, for $k = 2$, a good ordering is not necessarily required as long as the ordering contains a single good splitting point; however, since $k - 1$ good splitting points are generally required, it would seem more difficult to derive consistently good partitionings from a poor ordering.

We observe that there is some improvement by using $1 - 10$ eigenvectors as opposed to just 10 eigenvectors. This implies that using fewer than 10 eigenvectors can sometimes yield a better result than using 10 eigenvectors. This could result from several reasons: (i) inaccuracies in the clique hypergraph-to-graph model, (ii) poor choice of H for scaling eigenvectors, (iii) inability of our heuristic to best utilize the available information (probably due to its greedy behavior), and (iv) measuring the quality of a solution by a single Scaled Cost value (other objectives may lead to different conclusions). How to address these issues to better characterize the relationship between the number of eigenvectors used and the quality of the partitioning solution remains open.

Test Case	Scheme	Number of Clusters - k									Avg % improv
		10	9	8	7	6	5	4	3	2	
19ks	#1	9.18	8.65	8.05	7.57	6.65	6.01	5.41	5.02	4.79	-2.37
	#2	9.85	9.10	8.31	7.87	6.84	6.14	5.32	4.99	4.79	+0.00
	#3	9.62	8.99	8.20	7.82	7.83	6.16	5.28	4.97	4.79	+0.69
	#4	9.17	8.62	8.01	7.44	6.52	5.87	5.40	5.02	4.79	-3.14
bm1	#1	25.9	23.8	21.4	17.9	14.1	11.6	9.01	6.61	5.53	-1.89
	#2	25.5	23.4	21.8	18.6	16.0	12.5	8.63	6.61	5.53	+0.00
	#3	25.2	22.8	21.0	18.2	15.4	11.0	9.02	6.61	5.53	-2.31
	#4	26.2	24.1	21.8	18.3	14.6	12.5	9.01	6.61	5.53	-0.02
prim1	#1	46.1	44.0	42.2	38.6	35.8	31.8	27.4	20.5	13.4	+7.24
	#2	44.6	41.9	39.7	37.0	34.0	29.4	22.5	17.1	13.4	+0.00
	#3	41.2	38.6	36.2	33.4	29.8	25.9	22.5	17.1	13.4	-6.48
	#4	46.4	43.7	41.5	38.6	34.6	27.9	26.2	18.5	13.4	+3.84
prim2	#1	12.6	12.7	11.4	10.5	9.66	8.94	7.76	6.75	4.81	-2.96
	#2	13.7	12.7	12.0	11.2	10.1	9.18	7.95	6.76	4.71	+0.00
	#3	12.7	12.1	11.5	10.7	10.3	9.21	8.31	6.82	4.72	-1.41
	#4	12.4	11.9	11.2	10.3	9.39	8.75	7.76	6.75	4.81	-4.74
test02	#1	21.4	20.1	18.8	17.6	16.7	15.5	14.1	11.2	8.23	+4.89
	#2	21.1	19.9	18.5	17.0	15.4	13.9	12.4	10.7	8.07	+0.00
	#3	21.8	21.0	19.9	18.6	16.7	15.2	12.3	10.8	8.07	+4.50
	#4	21.4	20.3	19.0	17.2	16.3	15.4	13.6	11.2	8.23	+4.18
test03	#1	19.6	19.9	18.6	17.1	14.9	14.0	12.8	11.5	9.14	+4.37
	#2	19.0	17.6	16.7	15.3	14.6	13.7	12.5	11.6	9.29	+0.00
	#3	18.3	17.3	16.9	15.2	14.4	13.5	12.2	11.2	9.29	-1.50
	#4	19.9	18.6	16.9	16.1	14.7	13.8	12.7	11.6	9.29	+2.11
test04	#1	13.6	12.6	11.9	11.2	10.3	9.59	8.89	7.57	5.78	+3.97
	#2	13.2	12.3	11.5	10.8	9.97	9.32	8.21	6.83	5.78	+0.00
	#3	13.4	12.4	11.8	11.0	10.2	9.24	8.43	6.83	5.78	+1.18
	#4	13.6	12.7	11.9	11.2	10.3	9.59	8.89	7.56	5.78	+4.04
test05	#1	7.18	6.94	6.66	6.38	5.96	5.58	5.13	4.77	3.09	+2.23
	#2	7.42	7.03	6.53	6.11	5.79	5.50	4.85	4.35	3.09	+0.00
	#3	7.45	7.15	6.68	6.28	5.91	5.62	5.14	4.77	3.09	+2.85
	#4	7.77	7.00	6.74	6.39	5.91	5.60	4.85	4.35	3.09	+1.71
test06	#1	24.4	19.3	17.9	16.3	14.6	12.9	11.6	9.96	8.80	+0.41
	#2	21.3	20.2	18.5	16.7	14.7	13.5	11.3	9.54	8.80	+0.00
	#3	20.7	19.7	18.4	17.0	14.8	13.0	11.3	9.47	8.80	-0.87
	#4	22.2	20.9	19.3	17.7	15.9	14.5	12.5	10.3	8.80	+5.40
balu	#2	54.0	50.1	46.5	43.2	40.0	36.7	32.3	24.4	17.6	+0.00
	#3	53.2	49.1	44.2	42.4	40.0	36.7	32.3	24.8	17.6	-0.96
struct	#2	12.9	12.0	10.9	9.82	8.46	7.56	6.53	5.54	4.25	+0.00
	#3	12.9	12.0	10.9	9.87	8.94	7.86	6.90	5.83	4.62	+3.12
biomed	#2	1.87	1.73	1.62	1.49	1.34	1.23	1.11	0.89	0.61	+0.00
	#3	1.97	1.86	1.72	1.53	1.39	1.26	1.09	0.88	0.61	+2.61
Average #2 vs	#1	+0.62	+1.14	+1.04	+0.99	-0.08	+0.78	+5.71	+4.83	+0.26	+1.70
	#3	-1.32	-0.58	-0.30	-0.09	+1.35	-1.08	+1.37	+0.95	+0.69	+0.01
	#4	+0.68	+0.91	+0.88	+0.86	-0.15	+0.79	+5.00	+3.28	+0.45	+1.41

Table 2: Scaled Cost ($\times 10^{-5}$) values for MELO orderings for four different weighting schemes. Scheme #1 uses $H = \infty$; #2 uses $H = \lambda_2 + \lambda_d$; #3 scales each eigenvector by dividing by its eigenvalue; and #4 adjusts H dynamically so that the sum of the unused eigenvector contributions is zero. The rightmost column gives the percentage improvements of scheme #2 versus the other schemes for each benchmark, and the bottom rows give the percent improvement as a function of k.

Table 4 reports results for our third set of experiments, which compare MELO to the multi-way partitioning algorithms RSB (recursive spectral bipartitioning) [25], KP [10], and SFC [1]. The codes for RSB

Test Case	d	Number of Clusters - k									Avg %
		10	9	8	7	6	5	4	3	2	improv
19ks	$1-10$	9.85	9.10	8.31	7.87	6.84	6.14	5.32	4.99	4.79	+0.00
	1	15.1	14.3	13.8	13.2	12.2	11.1	9.51	7.49	6.25	+37.9
	10	9.85	9.23	8.45	8.25	7.33	6.82	6.14	5.86	5.20	+6.71
bm1	$1-10$	25.5	23.4	21.8	18.6	16.0	12.5	8.63	6.61	5.53	+0.00
	1	29.6	26.2	23.9	20.8	16.7	12.5	9.02	6.61	5.53	+5.82
	10	27.6	25.2	22.7	19.4	16.0	12.7	8.86	6.90	6.10	+4.51
prim1	$1-10$	44.6	41.9	39.7	37.0	34.0	29.4	22.5	17.1	13.4	+0.00
	1	80.4	75.4	68.7	61.6	54.9	47.1	38.4	31.7	13.4	+37.1
	10	44.6	41.9	39.7	37.1	34.0	30.3	24.3	22.2	20.5	+7.58
prim2	$1-10$	13.7	12.7	12.0	11.2	10.1	9.18	7.95	6.76	4.71	+0.00
	1	19.6	18.2	16.8	15.4	13.5	11.2	9.45	7.18	5.55	+21.8
	10	13.7	13.1	12.6	11.9	11.0	10.2	8.87	7.23	4.71	+5.42
test02	$1-10$	21.1	19.9	18.5	17.0	15.4	13.9	12.4	10.7	8.07	+0.00
	1	24.4	31.9	29.6	27.8	25.5	22.4	19.7	16.0	12.1	+34.3
	10	21.2	20.0	18.7	17.2	15.5	13.9	12.6	11.8	9.48	+3.29
test03	$1-10$	19.0	17.6	16.7	15.3	14.6	13.7	12.5	11.6	9.29	+0.00
	1	30.7	28.3	25.4	23.1	21.5	19.3	16.5	14.9	11.8	+30.3
	10	19.0	17.6	16.9	15.3	14.6	13.7	12.5	11.6	9.45	+0.32
test04	$1-10$	13.2	12.3	11.5	10.8	9.97	9.32	8.21	6.83	5.78	+0.00
	1	37.9	34.3	30.2	25.1	19.6	15.4	11.6	8.45	5.78	+42.8
	10	13.2	12.3	11.5	10.9	10.2	9.37	8.21	6.83	5.93	+0.69
test05	$1-10$	7.42	7.03	6.53	6.11	5.79	5.50	4.85	4.35	3.09	+0.00
	1	11.7	10.8	9.60	8.78	7.96	7.04	6.32	4.95	3.09	+24.3
	10	8.30	7.84	7.48	7.02	6.46	5.82	5.54	4.95	4.04	+12.3
test06	$1-10$	21.3	20.2	18.5	16.7	14.7	13.5	11.3	9.54	8.80	+0.00
	1	31.7	29.5	27.0	24.2	22.9	21.5	19.2	15.6	14.3	+35.4
	10	32.0	28.9	25.0	22.2	21.1	18.7	16.9	14.9	13.8	+30.9
balu	$1-10$	54.0	50.1	46.5	43.2	40.0	36.7	32.3	24.4	17.6	+0.00
	1	95.3	91.3	86.1	79.3	70.2	62.2	60.9	55.7	47.8	+47.8
	10	54.0	50.1	46.5	43.2	40.0	36.7	32.3	24.9	17.8	+0.35
struct	$1-10$	12.9	12.0	10.9	9.82	8.46	7.56	6.53	5.54	4.25	+0.00
	1	18.8	17.0	15.3	13.8	12.2	10.7	8.72	6.74	4.85	+26.0
	10	12.9	12.0	10.9	9.82	8.46	7.56	6.53	5.54	4.25	+0.00
biomed	$1-10$	1.87	1.73	1.62	1.49	1.34	1.23	1.11	0.89	0.61	+0.00
	1	4.82	4.43	4.02	3.52	2.96	2.33	1.54	1.30	0.85	+47.7
	10	2.07	1.91	1.73	1.57	1.42	1.31	1.16	0.89	0.61	+5.18
Average vs $1-10$	1	+37.1	+38.6	+37.6	+36.8	+35.3	+32.0	+30.0	+26.4	+19.6	+32.6
	10	+5.15	+5.16	+4.81	+4.98	+5.34	+5.37	+7.10	+9.00	+11.0	+6.43

Table 3: Scaled cost values ($\times 10^{-5}$) for MELO orderings: $d = 1$ uses the ordering induced by sorting the entries of the Fiedler vector; $d = 10$ uses $\vec{\mu}_2, \vec{\mu}_3, \ldots \vec{\mu}_{11}$; and $d = 1 - 10$ reports the best values from using between 1 and 10 eigenvectors.

and KP were obtained from Dr. Pak Chan; however, the results reported in Table 4 are considerably better than the values reported in [10].[3] RSB constructs ratio cut bipartitionings by choosing the best of

[3]For the experiments performed in [10], nets with more than 99 pins were removed before the eigenvectors were computed using the Frankle clique model. For some of the circuit netlists (test03, test04, and test06), removing large nets disconnected the graph, forcing $\lambda_2 = 0$. Since RSB uses $\vec{\mu}_2$ to determine its initial ordering, and since eigenvectors with 0 as an eigenvalue are degenerate, the RSB results were worsened significantly by removing the large nets. In addition, results for KP also suffered. Hence, we ran both RSB and KP using the Frankle and partitioning-specific net models for the seven benchmarks used in [10] (prim1-2, test02-6). We observed that: (i) for RSB, the partitioning-specific net model averaged 0.19% and 11.1% respective improvement over the Frankle net model and the results from [10]; (ii) for KP, the Frankle model averaged 2.97% and 2.96% respective improvement over the partitioning-specific model and the results from [10]. Hence, in Table 4 results are given for the partitioning-specific model for RSB and the Frankle model for KP. Note that for MELO, only experiments

all splits of the Fiedler vector, and the algorithm is iteratively applied to the largest remaining cluster. The SFC and DP-RP codes were acquired from their authors [1]; experiments for SFC were done using the partitioning-specific net model. The results for all four algorithms are given in Table 4. MELO averaged 10.6%, 15.8% and 13.2% improvement over RSB, KP, and SFC respectively. MELO performed very consistently, outperforming the other algorithms in 32 of 39 head-to-head benchmark comparisons (see the rightmost column of the Table) and over the entire range of k.

Our final set of experiments used MELO orderings to construct bipartitionings by choosing the one with lowest ratio cut from all possible splits of the ordering, while ensuring that each cluster contains at least 45% of the modules. We quote the PARABOLI results of [38] as a source of comparison, and additionally compare against SB.[4] The MELO results reported are the best observed from splitting each of the ten orderings constructed for schemes #2, #3, and #4; we use three schemes since the best scheme for this application remains unclear (in fact, frequently one scheme performed best for one benchmark and worst for another). MELO ratio cuts average 42.0% and -2.93% improvement over SB and PARABOLI respectively (however, it is also the case that PARABOLI averages -6.23% improvement over MELO). Table 5 also reports the runtimes required for MELO to construct and split orderings using two and ten eigenvectors, after the eigenvectors have been computed. Despite MELO's $O(dn^2)$ complexity, the runtimes are quite reasonable because the algorithm is so simple (see [2] for detailed runtimes for eigenvector computations). The results from Table 5 do not immediately imply that MELO is a superior bipartitioner; rather, the results show that multiple-eigenvector based orderings can be used to yield high-quality balanced bipartitionings. We hypothesize that addressing the vector bipartitioning problem more directly will allow these results to be improved.

6 Conclusion and Future Work

We have proposed a new *vector partitioning* formulation and shown how to transform a graph partitioning instance into a vector partitioning instance. When $d = n$, our formulation is exactly equivalent to graph partitioning. We have also proposed MELO, a simple linear ordering algorithm that extends SB to multiple eigenvectors, and have found that MELO orderings lead to high-quality two-way and multi-way partitionings. We note many possible directions for future research:

- Modify MELO to run in sub-$O(n^2)$ time. During each iteration, MELO considers adding every possible vector to S. Alternatively, after the first vector is chosen, we could rank all the other vectors by their magnitude when added to the first vector. Then, instead of considering all possible vectors during an iteration, consider only vectors in a candidate set T of fixed size (e.g., 50), where T is constructed from the highest ranked vectors. Each time a vector chosen from T is added to S, the next highly

with the partitioning-specific net model were performed.

[4]The circuits s9234, s13207, and s15850 contain multiple connected components, so we ran MELO and SB on the largest component (between 97-99% of the total number of vertices) and added the remaining components to the smaller cluster, while ensuring that the 45% constraint was satisfied. Thus, the SB results differ from those reported in [38] for this reason and also because of net model differences.

ranked vector not in S or T is added to T. The remaining vectors are re-ranked periodically (e.g., every 100 iterations) and T is updated. We believe that this speedup will not adversely affect MELO's performance.

- Apply vector partitioning to clustering. By Corollary 6, a subset of vectors of relatively large magnitude that point in the same direction will correspond to a cluster of vertices with small cut. Thus, it should be possible to identify such subsets of vectors and thereby construct high-quality clusterings.

- Perform diagonal optimization. Cullum et al. [12] showed how to find a diagonal matrix D that minimizes the sum of the first d eigenvalues of $A + D$. Another possibility is to use the algorithm of Carter [9] to construct D such that $trace(D)$ is minimized. Many researchers, e.g., [8] [16] [37] [17], have applied various types of diagonal optimization both to improve the quality of lower bounds on the cost of a solution and to enhance the performance of their spectral partitioning heuristics.

  We believe that diagonal optimization would aid vector partitioning, because the d optimized eigenvectors would better approximate the exact n-dimensional vector partitioning instance than the d unoptimized vectors.

- Find and apply lower bounds. Graph spectra have been very useful for constructing lower bounds on partitioning, e.g., λ_2 is a lower bound on the cut of any bipartitioning. We would like to answer such questions as, "Assume that we have constructed a d-dimensional vector partitioning instance, and that we have an optimal solution for this instance. Can we guarantee the quality of the corresponding graph partitioning solution in terms of d?" The answer should help us understand how large d needs to be, i.e., for a given graph partitioning instance is there some d such that the vector partitioning instance is "close enough" to be sufficient?

- Finally, and most importantly, we would like to explore possible vector partitioning heuristics. The promising performance of MELO suggests that other vector partitioning heuristics will be successful. For example, an iterative local improvement method could be applied to 2-way vector partitioning.

Acknowledgements

We thank Pak Chan, Martine Schlag and Jason Zien for giving us their LASO2, KP, and RSB codes along with their entire set of experimental data. Thanks also go to Jon Frankle for his useful discussions and insight. Finally, we would like to thank the reviewers, especially Reviewer #1, for their extraordinarily thorough and helpful comments.

References

[1] C. J. Alpert and A. B. Kahng, Multi-way partitioning via spacefilling curves and dynamic programming, in: Proceedings of the ACM/IEEE Design Automation Conference (1994) 652-657.

[2] C. J. Alpert and A. B. Kahng, Multi-way partitioning via geometric embeddings, orderings, and dynamic programming, IEEE Trans. on CAD 14 (1995) 1342-1358.

[3] C. J. Alpert and A. B. Kahng, A general framework for vertex orderings, with applications to netlist clustering, in: IEEE International Conference on Computer-Aided Design (1994) 63-67.

[4] C. J. Alpert and A. B. Kahng, Recent developments in netlist partitioning: a survey, Integration: the VLSI Journal 19 (1995) 1-81.

[5] K. S. Arun and V. B. Rao, New heuristics and lower bounds for graph partitioning, in: Proceedings of the IEEE International Symposium on Circuits and Systems (1991) 1172-1175.

[6] S. T. Barnard and H. D. Simon, A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems, Concurrency: Practice and Experience 6 (1994) 101-117.

[7] E. R. Barnes, An algorithm for partitioning the nodes of a graph, Siam J. Alg. Disc. Methods 3 (1982) 541-549.

[8] E. R. Barnes, A. Vannelli, and J. Q. Walker, An new heuristic for partitioning the nodes of a graph, Siam J. Dicrete Mathematics 3 (1988) 299-305.

[9] M. Carter, The indefinite zero-one quadratic problem, Discrete Applied Mathematics 7 (1984) 23-44.

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

[11] H. R. Charney and D. L. Plato, Efficient partitioning of components, in: Proceedings of the Fifth Design Automation Workshop (1968) 16–0 - 16–21.

[12] J. Cullum, W. E. Donath and P. Wolfe, The minimization of certain nondifferentiable sums of eigenvalues of symmetric matrices, Mathematical Programming Study 3 (North-Holland, Amsterdam, 1975) 35-55.

[13] C. Delorme and S. Poljak, The performance of an eigenvalue bound on the max-cut problem in some classes of graphs, Discrete Mathematics 111 (1991) 145-156.

[14] C. Delorme and S. Poljak, Laplacian eigenvalues and the maximum cut problem, Mathematical Programming 62 (1993) 557-574.

[15] W. E. Donath, Logic partitioning, in: Physical Design Automation of VLSI Systems (Benjamin/Cummings 1988) 65-86.

[16] W. E. Donath and A. J. Hoffman, Lower bounds for the partitioning of graphs, IBM Journal of Research and Development 17 (1973) 420-425.

[17] J. Falkner, F. Rendl, and H. Wolkowicz, A computational study of graph partitioning, Mathematical Programming 66 (1994) 211-239.

[18] M. Fiedler, Algebraic connectivity of graphs, Czechoslovak Mathematical Journal 23 (1973) 298-305.

[19] J. Frankle and R. M. Karp, Circuit placements and cost bounds by eigenvector decomposition, in: Proceedings of the IEEE International Conference on Computer-Aided Design (1986) 414-417.

[20] A. Frieze and M. Jerrum, Improved approximation algorithms for max k-cut and max bisection, in: Proceedings of the 4th International IPCO Conference 920 (1995) 1-13.

[21] J. Garbers, H. J. Promel and A. Steger, Finding clusters in VLSI circuits, in: Proceedings of the IEEE International Conference on Computer-Aided Design (1990) 520-523.

[22] M. X. Goemans and D. P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, Journal of the ACM 42 (1995) 1115-1145.

[23] S. Guattery and G. L. Miller, On the performance of spectral graph partitioning methods, in: Proceedings of the ACM/SIAM Symposium on Discrete Algorithms (1995) 233-242.

[24] L. Hagen and A. B. Kahng, A new approach to effective circuit clustering, in: Proceedings of the IEEE International Conference on Computer-Aided Design (1992) 422-427.

[25] L. Hagen and A. B. Kahng, New spectral methods for ratio cut partitioning and clustering, IEEE Trans. on CAD 11 (1992) 1074-1085.

[26] S. W. Hadley, B. L. Mark, and A. Vannelli, An efficient eigenvector approach for finding netlist partitions, IEEE Trans. on CAD 11 (1992) 885-892.

[27] K. M. Hall, An r-dimensional quadratic placement algorithm, Manag. Sci. 17 (1970) 219-229.

[28] M. Hanan, P. K. Wolff, and B. J. Agule, A study of placement techniques, Journal of Design Automation and Fault-Tolerant Computing 2 (1978) 28-61.

[29] B. Hendrickson and R. Leland, An improved spectral graph partitioning algorithm for mapping parallel computations, SIAM Journal on Scientific Computing 16 (1995) 452-69.

[30] T. C. Hu and K. Moerder, Multiterminal flows in a hypergraph, in: VLSI Circuit Layout: Theory and Design (IEEE Press, 1985) 87-93.

[31] E. Ihler, D. Wagner, and F. Wagner, Modeling hypergraphs by graphs with the same mincut properties, Information Processing Letters 45 (1993) 171-175.

[32] T. Lengauer, Combinatorial algorithms for integrated circuit layout, (Wiley-Teubner, 1990).

[33] B. Mohar, The Laplacian spectrum of graphs, in: Proceedings of the 6th Quadrennial International Conference on the Theory and Applications of Graphs (1988) 871-898.

[34] L. T. Pillage and R. A. Rohrer, A quadratic metric with a simple solution scheme for initial placement, in: Proceedings of the ACM/IEEE Design Automation Conference (1988) 324-329.

[35] S. Poljak and F. Rendl, Solving the max-cut problem using eigenvalues, Discrete Applied Mathematics 62 (1995) 249-278.

[36] A. Pothen, H. D. Simon, and K. P. Liou, Partitioning sparse matrices with eigenvectors of graphs, SIAM Journal of Matrix Analysis and its Applications 11 (1990) 430-452.

[37] F. Rendl and H. Wolkowicz, A projection technique for partitioning the nodes of a graph, Annals of Operations Research 58 (1995) 155-179.

[38] B. M. Riess, K. Doll, and F. M. Johannes, Partitioning very large circuits using analytical placement techniques, in: Proceedings of the ACM/IEEE Design Automation Conference (1994) 646-651.

[39] D. S. Scott, LASO2 documentation, Technical Report, University of Texas, Austin, TX (1980).

[40] D. A. Spielman and S.-H. Teng, Spectral partitioning works: planar graphs and finite element meshes, Manuscript (1996).

[41] R.-S. Tsay and E. S. Kuh, A unified approach to partitioning and placement, IEEE Transactions on Circuits and Systems 38 (1991) 521-533.

Test Case	Scheme	Number of Clusters - k									Avg %
		10	9	8	7	6	5	4	3	2	improv
19ks	MELO	9.85	9.10	8.31	7.87	6.84	6.14	5.32	4.99	4.79	+0.00
	RSB	13.9	11.6	9.15	8.74	8.87	7.00	6.51	6.45	6.35	+18.9
	KP	9.09	9.37	10.7	9.52	9.00	9.00	6.95	6.58	6.20	+17.8
	SFC	15.1	14.3	13.8	13.2	12.2	11.1	8.37	7.48	5.44	+35.7
bm1	MELO	25.5	23.4	21.8	18.6	16.0	12.5	8.63	6.61	5.53	+0.00
	RSB	33.3	31.1	26.9	22.7	17.0	11.3	8.63	6.61	5.53	+8.94
	KP	27.5	23.1	18.9	18.2	12.5	10.7	8.67	6.61	5.53	-4.97
	SFC	24.8	22.8	20.7	18.0	14.4	11.5	8.89	6.61	5.53	-3.17
prim1	MELO	44.6	41.9	39.7	37.0	34.0	29.4	22.5	17.1	13.4	+0.00
	RSB	53.1	44.6	40.5	38.1	32.4	28.1	23.9	17.0	13.4	+2.51
	KP	44.7	41.3	32.3	33.2	31.3	29.9	21.2	14.7	13.5	-6.16
	SFC	38.9	36.7	35.2	31.7	28.8	26.0	21.8	14.6	13.4	-10.6
prim2	MELO	13.7	12.7	12.0	11.2	10.1	9.18	7.95	6.76	4.71	+0.00
	RSB	11.2	10.9	10.1	9.73	9.33	8.33	7.85	7.69	5.55	-5.33
	KP	15.0	15.2	13.5	11.0	10.5	10.1	9.23	7.25	4.64	+7.38
	SFC	13.7	13.3	12.8	12.1	11.0	9.43	7.95	6.86	5.05	+4.14
test02	MELO	21.1	19.9	18.5	17.0	15.4	13.9	12.4	10.7	8.07	+0.00
	RSB	31.3	31.4	20.2	29.8	28.6	28.8	18.4	18.2	12.4	+36.4
	KP	24.4	22.6	22.1	19.1	19.3	18.7	17.4	12.0	9.26	+16.8
	SFC	25.5	24.1	22.8	20.9	18.5	16.1	13.4	10.9	8.07	+12.4
test03	MELO	19.0	17.6	16.7	15.3	14.6	13.7	12.5	11.6	9.29	+0.00
	RSB	20.3	19.9	17.7	17.0	16.7	17.6	16.0	14.3	11.9	+14.6
	KP	20.3	22.4	19.1	17.3	18.4	22.8	19.9	14.7	9.45	+19.2
	SFC	22.6	21.1	19.2	17.1	16.2	15.2	14.3	13.0	10.2	+12.0
test04	MELO	13.2	12.3	11.5	10.8	9.97	9.32	8.21	6.83	5.78	+0.00
	RSB	14.3	12.5	11.8	11.1	10.2	10.3	9.08	8.65	5.85	+6.46
	KP	18.3	16.3	15.0	12.2	13.7	12.5	8.98	12.0	6.74	+22.9
	SFC	22.2	19.9	17.8	17.6	16.5	15.1	11.6	8.19	5.78	+30.7
test05	MELO	7.42	7.03	6.53	6.11	5.79	5.50	4.85	4.35	3.09	+0.00
	RSB	7.50	6.82	6.70	6.12	5.38	4.97	4.26	4.06	3.09	-3.85
	KP	10.8	10.6	9.28	6.80	6.60	6.69	7.81	7.34	4.52	+27.2
	SFC	9.88	8.66	8.06	7.84	7.32	6.56	5.49	4.90	3.09	+16.1
test06	MELO	21.3	20.2	18.5	16.7	14.7	13.5	11.3	9.54	8.80	+0.00
	RSB	17.8	17.3	16.0	15.6	16.2	17.3	19.9	15.6	14.3	+10.3
	KP	21.0	20.9	19.7	18.5	19.1	19.1	18.0	18.2	28.6	+24.9
	SFC	27.1	25.1	23.7	20.2	18.4	16.5	13.7	11.3	9.21	+17.3
balu	MELO	54.0	50.1	46.5	43.2	40.0	36.7	32.3	24.4	17.6	+0.00
	RSB	72.2	76.6	81.8	91.3	84.6	74.5	57.8	55.4	47.2	+46.9
	KP	45.7	58.6	56.8	47.9	45.9	35.9	33.2	32.9	48.7	+14.2
	SFC	82.0	79.1	74.1	70.3	64.9	62.2	49.4	47.3	17.6	+34.3
struct	MELO	12.9	12.0	10.9	9.82	8.46	7.56	6.53	5.54	4.25	+0.00
	RSB	9.52	9.15	8.72	7.96	8.03	6.60	6.15	5.68	4.85	-10.7
	KP	14.9	15.7	13.8	14.4	11.4	9.32	7.91	7.51	6.60	+23.8
	SFC	12.1	11.2	10.5	9.41	8.65	7.93	7.05	6.42	4.85	+2.04
biomed	MELO	1.87	1.73	1.62	1.49	1.34	1.23	1.11	0.89	0.61	+0.00
	RSB	1.68	1.67	1.60	1.58	1.60	1.20	1.27	1.28	0.85	+8.29
	KP	3.22	3.03	2.65	1.75	1.63	1.36	1.83	1.16	0.84	+24.3
	SFC	1.84	1.69	1.59	1.47	1.51	1.48	1.25	1.15	0.85	+9.22
Average	RSB	+4.39	+5.47	+2.64	+8.30	+11.8	+9.60	+13.9	+19.7	+20.0	+10.6
	KP	+10.2	+16.0	+13.4	+9.35	+12.8	+15.8	+20.1	+21.1	+23.2	+15.8
	SFC	+13.6	+13.5	+14.0	+14.0	+15.2	+15.4	+14.0	+13.2	+6.06	+13.2

Table 4: Scaled cost ($\times 10^{-5}$) comparisons for partitionings derived from MELO orderings versus RSB [25], KP [10], and SFC [1] algorithms.

Test Case	SB		PARABOLI		MELO		Runtimes(s)		improv % vs SB	improv % vs PARABOLI
	cuts	RC	cuts	RC	cuts	RC	$d = 2$	$d = 10$		
19ks	179	8.92			119	5.89	40	79	+34.0	
bm1	75	38.9			48	30.0	4	9	+22.9	
prim1	75	43.6	53	30.6	64	36.9	3	8	+15.4	-17.1
prim2	254	11.3	146	6.47	169	7.48	26	89	+33.9	-13.6
test02	196	28.4			106	15.4	9	29	+46.6	
test03	85	13.2			60	9.29	9	27	+29.7	
test04	207	36.2			61	10.6	8	24	+70.8	
test05	167	9.93			102	6.07	20	67	+39.0	
test06	295	38.5			90	11.7	10	31	+69.7	
balu	110	69.2	41	25.8	28	17.6	3	7	+74.6	+31.8
struct	49	5.16	40	4.20	38	4.00	12	38	+22.5	+4.77
biomed	286	2.69	135	1.28	115	1.08	132	496	+59.9	+15.7
s9234	166	1.95	74	0.86	79	0.92	108	516	+52.8	-6.53
s13207	110	0.57	91	0.48	104	0.54	186	710	+5.27	-12.1
s15850	125	0.46	91	0.33	52	0.19	308	1197	+58.7	+42.4
industry2	525	1.32	193	0.49	319	0.81	478	1855	+38.6	-39.5

Table 5: Min-cut and ratio cut ($\times 10^{-5}$) comparisons for MELO derived bipartitionings versus PARABOLI [38] and SB [25]. PARABOLI results are quoted from [38]. All three algorithms require each cluster to contain at least 45% of the total modules. MELO runtimes are given for a Sun Sparc 10 and reported in seconds.