INFORMS Journal on Computing
Vol. 18, No. 2, Spring 2006, pp. 197–208
ISSN 1091-9856 | EISSN 1526-5528 | 06 | 1802 | 0197

DOI 10.1287/ijoc.1050.0154

Using Eigenvectors to Partition Circuits

Dorothy Kucar, Anthony Vannelli
Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, N2L 3G1 Canada
{dkucar@us.ibm.com, vannelli@cheetah.vlsi.uwaterloo.ca}

Many fields, ranging from bioinformatics to databases to large-scale integrated circuits, deal with the interrelation between elementary objects. Objects are represented as vertices and the relationships between them are represented as nets (edges that connect two or more vertices) in the form of a hypergraph. We seek a placement of vertices that groups like objects and separates unlike objects. This involves separating related objects into a few, possibly disjoint, blocks. These *hypergraph-partitioning* problems are NP-hard so cannot be solved exactly, except for very small instances. We develop and use a numerical technique based on eigenvector decomposition of the connectivity matrix associated with the circuit netlist to partition hypergraphs emanating from circuit netlists. The eigenvector components of the circuit connectivity matrix are then used to determine vertex coordinates in one dimension that are then rounded in some fashion to determine block assignments. The inherent difficulty with eigenvector techniques is that the eigenvector components tend to cluster, making it difficult to determine correct block assignments. Our technique uses weights on nets, vertices, and fixed vertices to obtain a more "discrete" placement of vertices, making it easier to determine correct block assignments.

Key words: hypergraph partitioning; eigenvectors; fixed terminals; one-dimensional vertex placement
History: Accepted by John W. Chinneck, Area Editor for Modeling: Methods & Analysis; received October 2003; revised February 2005; accepted June 2005.

1. Introduction

Allocation of electronic switching units has many implications with respect to criteria such as routability, wiring congestion, and timing performance. When units with numerous wiring connections between themselves are placed in proximity to one another, these criteria are more likely to be met. A circuit is represented as a hypergraph where switching units are vertices weighted by the number of incoming wire segments and *signal nets* are edges with two or more endpoints. Additionally, circuits must connect to the outside world through input/output *terminals*. In this paper, a hypergraph is composed of $|V|$ vertices (some of which may be fixed terminals), $|E|$ nets, and k blocks.

Our goal is to divide $|V|$ vertices into k blocks such that the number of nets between the blocks, called the *net cut*, is minimized. The rationale behind this is that switching units (vertices) that perform related tasks have numerous connections between them and should be assigned to the same block; those that have few connections between them are assigned to different blocks. Techniques that use eigenvectors usually partition into two blocks, so in our results, k is assumed to be two. However, our technique is by no means limited to the case where $k = 2$. Recently, it has also become essential to include fixed terminals into the partitioning formulation (Alpert et al. 2000).

The problem we have just described appears at several stages of the very large scale integrated (VLSI) circuit design process and has generated a great deal of interest in the VLSI-CAD community over the last 30 years. Current partitioning techniques include multilevel interchange-based methods (Karypis et al. 1997) and methods that use eigenvectors of graphs to obtain vertex placements that are subsequently split in some fashion to yield block assignments (Hall 1970, Fiedler 1973, Pothen et al. 1990). Problems associated with using eigenvectors to obtain block assignments are as follows.

(a) The eigenvector components tend to cluster, which results in poor one-dimensional placements and, hence, partitions.

(b) Eigenvector methods work with graph representations of hypergraphs—any graph representation of a hypergraph is merely an *approximation* of the hypergraph.

(c) Eigenvector approaches are generally not amenable to inclusion of fixed vertices, so they cannot be made to model problems from circuit design very effectively.

With regard to (a), what is required, in essence, are means to make the eigenvector components more monotone. There are several ways of improving the monotonicity of eigenvector components:

(i) by linearly combining eigenvectors corresponding to several of the largest or smallest eigenvalues (Hall 1970, Frankle and Karp 1986, Chan et al. 1994, Alpert et al. 1999);

(ii) by assigning weights to vertices (and not only nets);

(iii) by including fixed vertices in the formulation (Riess et al. 1994, Hendrickson et al. 1996).

We focus on the theoretical development of a unique eigenvector-based method based on points (ii) and (iii) to obtain better placements. We avoid linearly combining eigenvectors as in Alpert et al. (1999) due to the high computational cost associated with carrying multiple eigenvectors. Although we use a graph representation of a hypergraph, we assign weights to vertices and edges to approximate the hypergraph better than if we had used weights on edges alone. With respect to point (iii), our technique can be extended in a straightforward manner to handle fixed vertices.

Our method is based on a technique introduced by Otten (1982), which is used to order vertices into a one-dimensional placement. We formulate the eigenvector technique as an optimization problem where the solution is in the form of an eigenvector. The eigenvector, when sorted, is used to determine the relative row and column orderings of the incidence matrix of a hypergraph. We show how our technique may be used to determine k-way block assignments of hypergraphs. Furthermore, in practical applications, some of the vertices in the hypergraph may be fixed to specific blocks. We show how fixed vertices may be incorporated into the formulation. Results indicate that our technique yields significantly smaller net cuts than do two other well-studied eigenvector methods. If fixed vertices are incorporated into the formulation, then the net cut is improved by an additional 15% on average.

The rest of the paper is structured as follows. Section 2 covers previous eigenvector-partitioning approaches. Section 3 introduces a modified eigenvector technique that uses weights on vertices, nets, and fixed vertices to determine good vertex placements. Section 4 demonstrates how this technique can be used on a few simple examples. Section 5 contains results on circuit-test problems. Finally, §6 restates the problems associated with eigenvector techniques and summarizes how we solved these problems.

2. Previous Approaches

Given a graph $G(V, E)$, vertices $u \in V$ and $v \in V$, let $x_v = 1$ if vertex v belongs to block 1 and $x_v = -1$ if vertex v belongs to block 2. Let w_{uv} represent the weight of the edge connecting vertices u and v. We wish to minimize the number of edges with endpoints in both blocks. Since $x_v = \pm 1$, this is equivalent to minimizing the one-dimensional (integer) distance between all pairs of connected vertices (Tsay and Kuh 1991):

$$\min \sum_{(u,v)\in E} w_{uv}(x_u - x_v)^2. \tag{1}$$

The nonzero pattern in the summand results in the matrix formulation

$$\min_{\mathbf{x}} (\mathbf{P}^T\mathbf{x})^T\mathbf{W}(\mathbf{P}^T\mathbf{x}) = \min \mathbf{x}^T\mathbf{P}\mathbf{W}\mathbf{P}^T\mathbf{x} = \min \mathbf{x}^T\mathbf{L}\mathbf{x}, \tag{2}$$

where $\mathbf{P}$ is the $|V| \times |E|$ node-arc incidence matrix defined as

$$p_{v,e} = \begin{cases} +1 & \text{if vertex } v \text{ is the head of edge } e \\ -1 & \text{if vertex } v \text{ is the tail of edge } e \\ 0 & \text{otherwise.} \end{cases}$$

$\mathbf{L}$ is the Laplacian matrix of the graph and $\mathbf{W}$ is a diagonal matrix containing edge weights. The graph-partitioning formulation includes block assignment constraints on the vertices (Hall 1970, Pothen et al. 1990):

$$\min_{\mathbf{x}} \ \mathbf{x}^T\mathbf{L}\mathbf{x}$$
$$\text{s.t. } x_v = \pm 1.$$

Due to its discrete nature, this problem is very difficult to solve exactly for instances larger than several thousand vertices. In practice, the integer constraints are approximated by constraints that spread vertices about the median (constraints (4)) so there are an approximately equal number of vertices on both sides of the median (constraints (5)). For convenience we define $\mathbf{e}$ as the vector of all ones. Thus, the optimization problem is

$$\min_{\mathbf{x}} \ \mathbf{x}^T\mathbf{L}\mathbf{x} \tag{3}$$
$$\text{s.t. } \mathbf{x}^T\mathbf{x} = 1 \tag{4}$$
$$\mathbf{x}^T\mathbf{e} = 0. \tag{5}$$

This formulation essentially replaces the solution space consisting of the vertices of the ± 1 unit hypercube with the points on the surface of the Euclidean unit sphere.

THEOREM 1. *A globally optimal solution to* (3)–(5) *is the eigenvector corresponding to the second smallest eigenvalue* (λ_2) *of the Laplacian* (*Hall* 1970, *Fiedler* 1973).

The eigenvector corresponding to λ_2 is known as the *Fiedler vector* and is denoted by $\mathbf{u}_2$. The components of the Fiedler vector that are negative-valued represent the coordinates of vertices in the first block; nonnegative components of the Fiedler vector represent the coordinates of vertices in the second block. The effect is that the eigenvector components of strongly (weakly) connected vertices are close (far away), so strongly connected vertices are more likely to be assigned to the same block. Since it minimizes the distance between pairs of vertices, the technique we have just described can be used as a one-dimensional placement of vertices (Tsay and Kuh 1991). In §3, we use a better one-dimensional placement of vertices to determine a partition of vertices.

The components of $\mathbf{u}_2$ are sorted into ascending order to yield $\mathbf{u}_2^{\text{opt}}$. The coordinates of $\mathbf{u}_2^{\text{opt}}$ with

respect to $\mathbf{u}_2$ determine the ordering of the vertices. The net cut is the smallest cut between columns $[0.45|V|]$ and $[0.55|V|]$ of $\mathbf{P}$.

In the classical Fiedler optimization problem, the solution is a vector of vertex coordinates from -1 to 1. Fixed vertices can be modelled as vertices lying at the extreme ends of the $[-1, 1]$ range. Hendrickson et al. (1996) extend the Fiedler formulation to include fixed vertex information as follows:

$$\min \ \mathbf{x}^T\mathbf{L}\mathbf{x} - \tfrac{1}{2}\mathbf{g}^T\mathbf{x} \tag{6}$$

$$\text{s.t.} \ \mathbf{x}^T\mathbf{x} = 1 \tag{7}$$

$$\mathbf{x}^T\mathbf{e} = 0, \tag{8}$$

where $g_i > 0$ denotes a *preference* for vertex i toward the block corresponding to $+1$. This is an *extended eigenproblem* for which, although it has multiple solutions, the solution that minimizes the objective function is always the $\mathbf{x}$ vector associated with the smallest possible value of λ that satisfies the constraints.

Barnes' method tries to find a partition-assignment matrix that "best approximates" an adjacency matrix and thus determines how vertices should be assigned to blocks that minimize the number of edges that are cut (Barnes 1982). He uses the following definition.

Definition 1. A block adjacency matrix is defined as

$$b_{ij} = \begin{cases} 1 & \text{if vertex } i \text{ is in the same block as vertex } j \\ 0 & \text{otherwise.} \end{cases}$$

The eigenvalues of $\mathbf{B}$ are $\{\beta_1, \beta_2, \ldots, \beta_k, 0, \ldots, 0\}$.

Barnes (1982) restates k-way graph partitioning in terms of finding a block-assignment matrix $\mathbf{B}$ so that the distance (in a two-norm sense) between $\mathbf{B}$ and the adjacency matrix $\mathbf{A}$ is as small as possible. The rationale is that if vertices i and j are adjacent (i.e., $a_{ij} = 1$) then they should end up in the same block (i.e., $b_{ij} = 1$). In mathematical terms we would like to find $\mathbf{B}$ such that

$$\begin{aligned} \|\mathbf{A} - \mathbf{B}\|^2 &= \|\mathbf{A}\|^2 - 2\mathbf{A}\mathbf{B} + \|\mathbf{B}\|^2 \\ &= \sum_{i=1}^{|V|} \alpha_i^2 - 4(|E| - |E^c|) + \sum_{i=1}^{k} \beta_i^2 \end{aligned}$$

is as small as possible. Since the sums of eigenvalues and $|E|$ are known constants,

$$\min \|\mathbf{A} - \mathbf{B}\|^2 \equiv \min |E^c|.$$

We consider the k orthonormal eigenvectors $\mathbf{u}_j$ of $\mathbf{A}$ corresponding to the k largest eigenvalues of $\mathbf{A}$, $\alpha_1 \geq \cdots \geq \alpha_k$. For bipartitioning, where u_{i1} and u_{i2} are the ith components of the first and second eigenvectors of $\mathbf{A}$, m_1 and m_2 are the respective sizes of the two blocks. The optimal partition assignment is attained when the components

$$\bar{a}_i = \frac{u_{i1}}{\sqrt{m_1}} - \frac{u_{i2}}{\sqrt{m_2}}$$

are sorted from smallest to largest. Since eigenvectors can be positive or negative, Barnes suggests selecting the largest, in magnitude, of the four possible $\pm$ combinations of the two eigenvectors. We use the sorted order of $\bar{a}_i$ to order the adjacency matrix and compute the cut position between rows $\lfloor 0.45\ \mathrm{m} \rfloor$ and $\lfloor 0.55\ \mathrm{m} \rfloor$ of $\mathbf{P}$.

Some authors propose using several eigenvectors to determine one-dimensional placements of vertices that minimize the number of cuts. The reasoning behind using multiple eigenvectors (instead of just the second eigenvector) is that the subsequent eigenvectors provide additional information regarding where vertices should be assigned to minimize the net cut. Frankle and Karp (1986) and Alpert et al. (1999) divide the set of vertices into groups such that the two norm of the sum of the vertex coordinates in each group is maximized. Chan et al. (1994) use the cosine of the angle between two rows of the $|V| \times k$ eigenvector matrix $\mathbf{V}$ to determine how close the vertices are to each other. If the cosine between two vectors is close to 1, then the corresponding vertices must belong to the same block. Their k-way partitioning heuristic consists of constructing k prototype vectors with distinct directions (to represent blocks) and placing into the corresponding block the vertices that have corresponding vectors within $\pi/8$ radians of the prototype vector.

Riess et al. (1994) use the Fiedler vector to fix the vertices corresponding to the ten smallest eigenvector components and ten largest eigenvector components to locations 1.0 and 0.0, respectively. The center of gravity of the remaining vertices is fixed at location 0.5. They solve a quadratic program to reposition the free vertices so the overall wire length is reduced. In the next pass of the algorithm, the 5% of vertices with the largest (smallest) resulting coordinate are moved so their center of gravity is at $x_i = 0.95$ ($x_i = 0.05$). After performing the optimization and repositioning, the process is repeated at the center of gravity of $x_i = 0.9$ and $x_i = 0.1$, and so on. The process is repeated ten times so there are ten different placements, and the placement that gives the best partition is selected.

Since eigenvector methods work with Laplacian matrices of *graphs*, it is necessary to represent hypergraphs as equivalent graphs. Often, a hypergraph is represented as clique graph. A *clique* is a subgraph of graph $G(V, E)$ in which every vertex is adjacent to every other vertex, so a net on $|e|$ vertices will be represented by $\binom{|e|}{2}$ edges. In order that large nets do not have a disproportionate influence in the

determination of the net cut, graph edges of the net are weighted so that, regardless of how vertices in the graph are partitioned, the number of cuts should come as close as possible to 1. Unfortunately, there exists no weighting scheme that models the cut properties of nets with more than three vertices exactly, so any graph representation of a hypergraph is merely an approximation (Ihler et al. 1993).

3. A Weighted-Vertex-and-Edge Model

The problem we address in this paper is how to assign vertices to coordinates on a horizontal axis from -1 to $+1$ so that the number of nets that have vertices with both positive and negative coordinates is minimized *given* that both nets and vertices have nonunit weights. In effect, our formulation minimizes the overall connection distance between all pairs of vertices and is thus a one-dimensional *placement* of vertices.

The distinguishing feature that will allow us to obtain good-quality placements is weighting vertices according to their vertex degree (i.e., switching unit size) into the formulation. Since the target application of our work is VLSI physical design in which switching units *of varying sizes* are represented as vertices, the abstraction we present in this section models the vertex-placement problem better than previous approaches. Our model is formulated as a quadratic objective function with quadratic constraints, of which the solution is in the form of an eigenvector.

Recall that the Fiedler vector is an approximate solution to the problem that tries to map vertices to discrete locations $x_i = \pm 1$. However, the components of the Fiedler vector will tend to cluster tightly about the origin because it is minimizing over a hypersphere defined by $\mathbf{x}^T\mathbf{x} = 1$. In our formulation we spread out the ± 1 vector by minimizing over the hyperellipse with principal axes scaled by vertex weights. The resulting eigenvector components are reasonably spread out and can be used to order vertices.

The basis of this method is a direct representation of a hypergraph through a *vertex-edge incidence matrix* where edges may connect more than two vertices. Let $m = |V|$ be the number of vertices and let $n = |E|$ be the number of nets. Define an $m \times n$ vertex-edge incidence matrix $\mathbf{P}$ where

$$p_{ij} = \begin{cases} 1 & \text{vertex } i \text{ is connected to net } j \\ 0 & \text{otherwise.} \end{cases}$$

In order to obtain a good partitioning solution, we transform the incidence matrix into a form where the nonzeros in every row and column are consecutive, called a *perfect sequence*. The rationale behind this is that if an incidence matrix has the consecutive ones property in the rows and columns, then every vertex in the corresponding linear arrangement is placed next to only its immediate neighbors and the overall net length is minimized. In the example below, the rows and columns of an incidence matrix are permuted into the perfect-sequence form.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

When we compute the number of rows between the ones in each column, the matrix on the left has a total net length of $2+2+4+0+3 = 11$ units, whereas the perfect sequence matrix on the right has a total netlength of $0+1+1+1+1 = 4$ units.

Assume we have a hypergraph defined by a set of vertices and nets. We obtain a vertex placement by mapping vertices to the horizontal axis between -1 and $+1$ according to their weights multiplied by a sign factor (assuming vertex weights are greater than 0, but less than 1). Let r_i be the coordinate of vertex i and let c_{ii} be the weight of vertex i; then $r_i = \pm c_{ii}$. In other words, $c_{ii}^{-1} r_i = \pm 1$. Using the same technique as in §2, we approximate the constraints $c_{ii}^{-1} r_i = \pm 1$ by constraints $\sum_{i=1}^{m} c_{ii}^{-2} r_i^2 = 1$ and $\sum_{i=1}^{m} c_{ii}^{-2} r_i = 0$ where the second constraint indicates that the total weight of vertices on the left side of the origin is equal to the total weight of vertices on the right side of the origin. In VLSI terminology, where vertex weight corresponds to switching unit area, this corresponds to having an approximately equal area on either side of the origin.

Let d_{jj} be the inverse of the weight of net j and let w_{ik} be the weight of the *net segment* connecting vertices i and k:

$$w_{ik} = \sum_{j=1}^{n} p_{ij} d_{jj}^2 p_{kj}.$$

Now w_{ik} is the (i,k)th entry of the matrix $\mathbf{PD}^2\mathbf{P}^T$ where $\mathbf{P}$ is the incidence matrix and $\mathbf{D}$ is an $n \times n$ diagonal matrix with nonzero entries d_{jj}. The matrix $\mathbf{PD}^2\mathbf{P}^T$ is the summation of the weights of all net segments that are connected to vertices i and k. Note that $w_{ik} = w_{ki}$—we will use this property in our derivation.

The origin of the model presented here lies in a weighted distance-minimization problem that minimizes the overall distance between all pairs of *connected* vertices. This formulation is similar to the formulation in Hall (1970) except that it includes weights on net segments *and* vertices. The resulting new optimization-problem description of the

weighted vertex and edge model is

$$\min \sum_{i=1}^{m}\sum_{k=1}^{m} w_{ik}(r_i - r_k)^2 \tag{9}$$

$$\text{s.t.} \sum_{i=1}^{m}\left(\frac{r_i}{c_{ii}}\right)^2 = 1 \tag{10}$$

$$\sum_{i=1}^{m}\left(\frac{r_i}{c_{ii}}\right)\left(\frac{1}{c_{ii}}\right) = 0. \tag{11}$$

We define the weights on nets and vertices as

$$c_{ii} = \frac{1}{\sqrt{\sum_{j=1}^{n} p_{ij}}} \quad \text{and} \quad d_{jj} = \frac{1}{\sqrt{\sum_{k=1}^{m} p_{kj}}}.$$

Using the above weighting scheme, through some mathematical manipulation, the optimization problem (9)–(11) becomes

$$\max \sum_{i=1}^{m}\sum_{k=1}^{m} w_{ik} r_i r_k \tag{12}$$

$$\text{s.t.} \sum_{i=1}^{m}\left(\frac{r_i}{c_{ii}}\right)^2 = 1 \tag{13}$$

$$\sum_{i=1}^{m}\left(\frac{r_i}{c_{ii}}\right)\left(\frac{1}{c_{ii}}\right) = 0. \tag{14}$$

We can restate (12)–(14) in terms of matrices, where $\mathbf{C}$ and $\mathbf{D}$ are diagonal matrices containing vertex and net weights as defined previously:

$$\max_{\mathbf{r}} \ \mathbf{r}^T(\mathbf{PD})(\mathbf{PD})^T\mathbf{r} \tag{15}$$

$$\text{s.t.} \ (\mathbf{C}^{-1}\mathbf{r})^T(\mathbf{C}^{-1}\mathbf{r}) = 1 \tag{16}$$

$$(\mathbf{C}^{-1}\mathbf{r})^T(\mathbf{C}^{-1}\mathbf{e}) = 0. \tag{17}$$

Define matrix $\mathbf{F} = (\mathbf{CPD})(\mathbf{CPD})^T$, which is formed by pre- and post-multiplying the square matrix in the objective function (15) by the vertex-scaling matrix $\mathbf{C}$. The solution of (15)–(17) is the vector $\mathbf{r}$, which corresponds to an ordering of rows of the matrix $\mathbf{P}$ (i.e., vertices in the hypergraph). We also use a vector $\mathbf{k} = \mathbf{D}^2\mathbf{P}^T\mathbf{r}$, which corresponds to an ordering of the columns of $\mathbf{P}$ (i.e., nets in the hypergraph). To obtain an ordering of rows and columns of $\mathbf{P}$, we sort the components of $\mathbf{r}$ and $\mathbf{k}$ from smallest to largest components; call these sorted vectors $\mathbf{r}^{\text{opt}}$ and $\mathbf{k}^{\text{opt}}$, respectively. The coordinates of $\mathbf{r}^{\text{opt}}$ with respect to $\mathbf{r}$ and $\mathbf{k}^{\text{opt}}$ with respect to $\mathbf{k}$ give a good ordering of $\mathbf{P}$. The value of $\mathbf{r}$ is found according to

Theorem 2. *Let $\mathbf{u}_2$ be the eigenvector corresponding to the second-largest eigenvalue of the matrix $\mathbf{F}$. A globally optimal solution to* (15)–(17) *is* $\mathbf{r} = \mathbf{Cu}_2$.

A rigorous proof of the above theorem is fairly involved and is in the online supplement to this paper at http://joc.pubs.inform.org/. The following theorem shows that the largest eigenvalue of $\mathbf{F}$ is 1 and justifies why we use the eigenvector corresponding to the second eigenvalue of $\mathbf{F}$ instead of the eigenvector corresponding to the largest eigenvalue of $\mathbf{F}$.

Theorem 3. *The largest eigenvalue of*

$$\mathbf{F} = (\mathbf{CPD})(\mathbf{CPD})^T$$

is $\lambda_1 = 1$ *and the corresponding eigenvector is* $\mathbf{u}_1 = \mathbf{C}^{-1}\mathbf{e}/\|\mathbf{C}^{-1}\mathbf{e}\|$.

Proof. First, we show that $\mathbf{C}^2\mathbf{PD}^2\mathbf{P}^T$ has unit row sums i.e., $\mathbf{C}^2\mathbf{PD}^2\mathbf{P}^T\mathbf{e} = \mathbf{e}$, where $\mathbf{e}$ is the vector of all ones. Since $\mathbf{P}^T\mathbf{e}$ is a vector of column sums of $\mathbf{P}$ and $d_{jj}^2 = (\sum_{k=1}^{m} p_{kj})^{-1}$ and it follows that $\mathbf{D}^2\mathbf{P}^T\mathbf{e} = \mathbf{e}$. Furthermore, $\mathbf{Pe}$ is a vector of row sums of $\mathbf{P}$ and $c_{ii}^2 = (\sum_{j=1}^{n} p_{ij})^{-1}$, and it follows that $\mathbf{C}^2\mathbf{Pe} = \mathbf{e}$. Hence, $\mathbf{C}^2\mathbf{PD}^2\mathbf{P}^T\mathbf{e} = \mathbf{e}$. Note that $\mathbf{C}^2\mathbf{PD}^2\mathbf{P}^T$ is a row stochastic matrix. The largest eigenvalue of $\mathbf{C}^2\mathbf{PD}^2\mathbf{P}^T$ is $\lambda_1 = 1$ with corresponding eigenvector $\mathbf{e}$. Therefore, $\mathbf{CPD}^2\mathbf{P}^T\mathbf{CC}^{-1}\mathbf{e} = \mathbf{C}^{-1}\mathbf{e}$. Normalizing the eigenvector by letting $\mathbf{u}_1 = \mathbf{C}^{-1}\mathbf{e}/\|\mathbf{C}^{-1}\mathbf{e}\|$, we have shown that $(\mathbf{CPD})(\mathbf{CPD})^T\mathbf{u}_1 = \mathbf{u}_1$. Note that all eigenvalues/eigenvectors of $\mathbf{C}^2\mathbf{PD}^2\mathbf{P}^T$ satisfy $(\mathbf{C}^2\mathbf{PD}^2\mathbf{P}^T)\mathbf{u}_i = \lambda_i\mathbf{u}_i$. Then $\mathbf{CPD}^2\mathbf{P}^T\mathbf{CC}^{-1}\mathbf{u}_i = \lambda_i\mathbf{C}^{-1}\mathbf{u}_i$. Thus, the corresponding eigenvalues of $(\mathbf{CPD})(\mathbf{CPD})^T$ are the same as $\mathbf{C}^2\mathbf{PD}^2\mathbf{P}^T$ and the corresponding eigenvectors of $(\mathbf{CPD})(\mathbf{CPD})^T$ are $\mathbf{C}^{-1}\mathbf{u}_i$. Accordingly, λ_1 is the largest eigenvalue of $(\mathbf{CPD})(\mathbf{CPD})^T$ with corresponding eigenvector $\mathbf{u}_1 = \mathbf{C}^{-1}\mathbf{e}/\|\mathbf{C}^{-1}\mathbf{e}\|$. □

Since the largest eigenvalue of $\mathbf{F}$ corresponds to an eigenvector $\mathbf{u}_1 = \mathbf{C}^{-1}\mathbf{e}/\|\mathbf{C}^{-1}\mathbf{e}\|$, the placement of the vertices and nets of the hypergraph would be determined from the vectors

$$\mathbf{r} = \mathbf{Cu}_1 = \mathbf{C}\frac{\mathbf{C}^{-1}\mathbf{e}}{\|\mathbf{C}^{-1}\mathbf{e}\|} = \text{const} \cdot \mathbf{e}$$

$$\mathbf{k} = \mathbf{D}^2\mathbf{P}^T\mathbf{r} = \text{const} \cdot \mathbf{e},$$

which assigns the same coordinate to all vertices.

3.1. Incorporating Fixed Terminals into the Formulation

We use an analytical approach similar to Hendrickson et al. (1996) to incorporate fixed vertices into the formulation. Fixed vertices are connected to exactly one net so the vertex weight is $c_{ii} = 1$. Since we fix vertices to the right block using $r_i \approx 1$, it is also true that $r_i \approx r_i^2$. This approximation will enable us to solve the optimization problem using conventional rather than specialized eigenvector solvers. We modify (15) to read

$$\max_{\mathbf{r}} \mathbf{r}^T\mathbf{PD}^2\mathbf{P}^T\mathbf{r} + \mathbf{r}^T\mathbf{Gr}, \tag{18}$$

where $\mathbf{G}$ is diagonal with entry $g_{ii} = c$ if vertex i is fixed and c is a constant called the *fixed-terminal scaling parameter*. The value of c is experimentally determined in §5. The "+" sign in the objective function (18) is there because we are maximizing instead of minimizing as in (6).

3.2. Bipartitioning

In this section, we discuss what it means to partition an incidence matrix. Given a hypergraph and its incidence matrix $\mathbf{P}$, a balanced bipartitioning of the hypergraph corresponds to a decomposition of $\mathbf{P}$ into

$$\mathbf{P} = \left[\begin{array}{ccc} \mathbf{P}_1 & \mathbf{0} & \mathbf{N}_1 \\ \hline \mathbf{0} & \mathbf{P}_2 & \mathbf{N}_2 \end{array}\right]$$

where $\mathbf{P}_1$ and $\mathbf{P}_2$ have $\lfloor m/2 \rfloor$ and $\lceil m/2 \rceil$ rows, respectively. The horizontal line represents the *cut line*. Matrix $[\mathbf{N}_1 \ \mathbf{N}_2]^T$ has at least one nonzero in each column, so the number of columns of $[\mathbf{N}_1 \ \mathbf{N}_2]^T$ represents the *net cut*. In a perfect netlist, because each row and column has its nonzeros consecutively, the number of columns of $\mathbf{N}_1$ (i.e., the net cut) will tend to be small, although we do not make any claims regarding optimality.

3.3. Calculating the Net Cut

In this section, we examine how we might extract net-cut information from an ordering of the rows and columns of the incidence matrix $\mathbf{P}$. Recall that the rows of $\mathbf{P}$ correspond to vertices and the columns to nets. Thus, $\mathbf{P}$ is an abstraction of an arrangement of vertices where vertices are numbered from 1 to m according to their position in the linear arrangement. Given k blocks, algorithm `kway-net cut` in Figure 1 finds the smallest net cut for all vertices located between vertex a and vertex b. The algorithm then updates the net cut and removes the cutset from the set N. Since $b - a = 1/10(k-1)$, the complexity of this routine is given by $O((k-1)/10(k-1)) = O(1)$. Using this algorithm, it is possible to compute the number of cuts for

algorithm `kway-net cut`(netlist, k)
N set of all nets
V set of all vertices
net cut = 0
for $i = 1$ to $k-1$
  $a = \left\lceil \frac{|V|i}{k} - \frac{|V|}{20(k-1)} \right\rceil$, $b = \left\lceil \frac{|V|i}{k} + \frac{|V|}{20(k-1)} \right\rceil$
  for $j = a$ to b
    $R_j \leftarrow$ nets in N connecting vtx j, $j+1$
  end for
  $S_i \leftarrow \{R_j : |R_j| = \min |R_j|\}$
  net cut $\leftarrow |S_i|$ + net cut
  $N \leftarrow N - \{S_i\}$
end for
return net cut

Figure 1 Computing the *k*-Way Net Cut

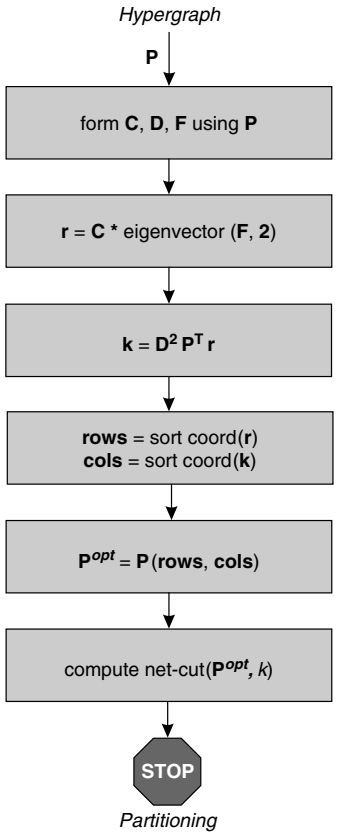

Figure 2 Eigenvector-Partitioning Technique

any integer number of partitions *without* recalculating eigenvectors. If one would like to partition the netlist into a different number of blocks, it is simply a matter of running the above algorithm again. The overall technique is in Figure 2.

4. Examples

The concepts presented above are illustrated with the aid of some simple examples. Our first example, borrowed from Otten (1982) is represented by the following incidence matrix:

$$\mathbf{P} = \begin{array}{c|ccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 3 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 5 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 6 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 7 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 8 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 9 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 10 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{array}$$

When we perform the eigenvector computation, we obtain unsorted row and column vectors:

$$\mathbf{r} = [-0.14, 0.02, 0.22, -0.71, -0.08, 0.18, -0.19, -0.48, 0.07, 0.35]$$

$$\mathbf{k} = [-0.10, 0.09, 0.04, 0.08, -0.04, 0.02, 0.12, -0.01, -0.47].$$

Sorting these from the smallest to largest components of $\mathbf{r}$ and $\mathbf{k}$ and ordering $\mathbf{P}$ accordingly yields

$$\mathbf{P}^{\text{opt}} = \begin{array}{c|cccccccccc} & 9 & 1 & 5 & 8 & 6 & 3 & 4 & 2 & 7 \\ 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 10 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}.$$

The placement before and after using our technique is in Figure 3.

Calculating the objective value yields $\mathbf{r}^T\mathbf{P}\mathbf{k} = (\mathbf{r}^{\text{opt}})^T\mathbf{P}^{\text{opt}}\mathbf{k}^{\text{opt}} = 0.84 = \lambda_2$ and the result is a perfect netlist. If we divide the vertices into two blocks such that $S_1 = \{4, 8, 7, 1, 5\}$ and $S_2 = \{2, 9, 6, 3, 10\}$, we would have to cut nets 6, 8, and 5. Alternatively, if we divide the vertices into three blocks $S_1 = \{4, 8, 7\}$, $S_2 = \{1, 5, 2, 9\}$, and $S_3 = \{6, 3, 10\}$, we would have to cut nets 3, 6, 5, and 1.

In our second example, we demonstrate that simply having monotone eigenvector components does not guarantee a perfect netlist. We can construct such an example by modifying row 3 of the matrix in the first example so $p_{3,5} = 1$:

$$\mathbf{P} = \begin{array}{c|ccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 3 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 5 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 6 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 7 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 8 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 9 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 10 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{array}.$$

The components of $\mathbf{r}^{\text{opt}}$ and $\mathbf{k}^{\text{opt}}$ are monotone:

$$\mathbf{r}^{\text{opt}} = [0.78, 0.50, 0.11, 0.07, 0.03, -0.06, -0.08, -0.13, -0.15, -0.26]$$

$$\mathbf{k}^{\text{opt}} = [0.64, 0.18, 0.00, -0.01, -0.08, -0.10, -0.18, -0.20, -0.26].$$

But $\mathbf{P}^{\text{opt}}$ is not a perfect netlist since $p^{\text{opt}}_{9,9} = 0$, but $p^{\text{opt}}_{8,9} = p^{\text{opt}}_{10,9} = 1$:

$$\mathbf{P}^{\text{opt}} = \begin{array}{c|cccccccccc} & 9 & 1 & 5 & 8 & 6 & 3 & 4 & 2 & 7 \\ 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}.$$

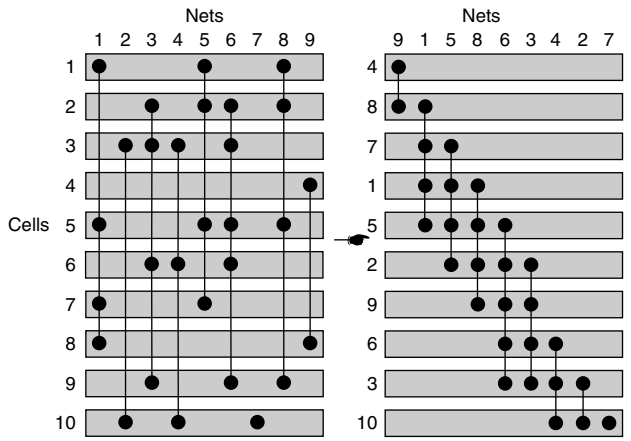

Figure 3 One-Dimensional Placement Before and After Ordering Using the Eigenvector Technique

On the other hand, for banded matrices,

$$\mathbf{P} = \mathbf{P}^{\text{opt}} = \begin{matrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{matrix}.$$

The eigenvector components follow a sinusoidal pattern

$$\mathbf{r}^{\text{opt}} = [-0.44, -0.39, -0.33, -0.21, -0.07, 0.07, 0.21, 0.33, 0.39, 0.44]$$

and the globally optimal net cut is exactly 4. In the last example, we fix the fifth vertex to the bottom block. If we add a weight of 1 to $f_{5,5}$, we obtain ordered vectors

$$\mathbf{r}^{\text{opt}} = [-0.53, -0.31, -0.29, -0.17, -0.14, 0.03, 0.06, 0.13, 0.38, 0.57]$$

$$\mathbf{k}^{\text{opt}} = [-0.53, -0.41, -0.38, -0.23, -0.16, -0.04, -0.04, 0.15, 0.47]$$

and the resulting matrix

$$\mathbf{P}^{\text{opt}} = \begin{matrix} & 7 & 2 & 4 & 3 & 6 & 5 & 8 & 1 & 9 \\ 10 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}.$$

The result reveals that vertex 5 is in the bottom block, as desired.

5. Experimental Results

We use the hypergraph (`.hgr`) and fixed vertex file (`.fix`) formats used by hMetis (Karypis and Kumar 1998). For the smaller test problems, we use our own `.hgr` and `.fix` files from test problems described in Kozminski (1991). We also use test problems described in Alpert (1998) and Caldwell et al. (1999). Our results were compiled on a 1.5 GB Pentium running Linux.

The most time-consuming part of our technique was computing the second eigenvector of **F**—we use the Lanczos method for finding a few eigenvectors and eigenvalues of this matrix. The Lanczos method converges to the outer eigenvalues and corresponding eigenvectors of the connectivity matrix very quickly (Watkins 1991, Demmel 1997). From the # steps column in Table 1, at most 30 iterations are required for all test problems—most problems require fewer than 15 iterations to converge. The time it takes the Lanczos routine to find the correct eigenvector depends on both the size of the problem and the number of Lanczos iterations involved. The first column in Figure 4 lists the test problems we use in this paper; the second and third columns are the number of vertices and the number of nets; the fourth column

Table 1 **Values of λ_2 for Test Problems**

Circuit	$\lvert V\rvert$	$\lvert E\rvert$	λ_2	# steps	Time
fract	149	163	0.98036659790	3	0.05
baluP	801	735	0.98203006224	3	0.18
prim1	833	902	0.99540420378	5	0.17
test	1,456	1,561	0.99065611868	3	0.26
struct	1,952	1,920	0.99961522379	7	0.34
ind1	2,271	2,192	0.99934016278	7	1.38
prim2	3,014	3,029	0.99326477632	5	0.5
bio	6,514	5,742	0.99955051973	8	1.52
ind2	12,637	13,419	0.99882199460	7	5.27
ind3	15,406	21,923	0.99853504321	9	3.58
avq.large	21,918	22,124	0.99997535474	28	11.24
avq.small	25,178	25,384	0.99997523342	29	13.53
golem3	103,048	144,949	0.99650732319	30	80.00
ibm01	12,752	14,111	0.99760048018	7	2.85
ibm02	19,601	19,584	0.99763290954	5	5.40
ibm03	23,136	27,401	0.99277058075	7	5.30
ibm04	27,507	31,970	0.99682436641	7	6.43
ibm05	29,347	28,446	0.98133068671	5	6.41
ibm06	32,498	34,826	0.99039151308	9	10.67
ibm07	45,926	48,117	0.99746066350	16	23.65
ibm08	51,309	50,513	0.99594016383	8	21.25
ibm09	53,395	60,902	0.99780986603	9	20.54
ibm10	69,429	75,196	0.99740936308	30	101.49
ibm11	70,558	81,454	0.99792244527	10	28.95
ibm12	71,076	77,240	0.99759478386	8	31.67
ibm13	84,199	99,666	0.99890526149	10	44.58
ibm14	147,605	152,772	0.99915583908	9	80.95
ibm15	161,570	186,608	0.99941107903	10	130.00
ibm16	183,484	190,048	0.99871341323	20	293.42
ibm17	185,495	189,581	0.99790119986	14	159.30
ibm18	210,613	201,920	0.99885952818	26	447.02

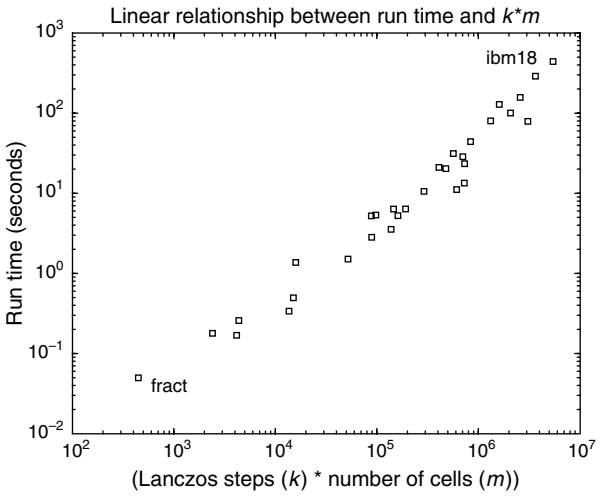

Figure 4 **Run Time Is Proportional to the Product of the Number of Vertices and the Number of Lanczos Steps Required to Converge to the Correct Eigenvalue**

contains the second eigenvalue of **F**; the fifth column contains the number of Lanczos steps required to compute the second eigenvalue; finally, the last column contains the time required to compute the second eigenvalue, in seconds.

In Figure 4, we observe an approximately linear relationship between run time and (# steps) × (# vertices).

5.1. Bipartitioning Results

The first experiment compares our eigenvector formulation "eig" to eigenvector formulations "Fiedler" described in Hall (1970) and to the Barnes (1982) method "Barnes." To reduce memory usage incurred in multiplying dense columns of **P** with dense rows of $\mathbf{P}^T$ in the computations to determine **L** and **F**, nets that connect more than 100 vertices are removed. The second and third columns of Table 2 contain hMetis results obtained using the following parameter settings (Karypis and Kumar 1998):

hMetis Parameter Settings

`Parts`	partition into 2 blocks
`Cofactor`	allow 10% imbalance
`Nuns`	20 runs
`Ectype`	coarsen type: modified hyperedge coarsening
`Retype`	refine type: FM
`Cycle`	no V–cycling
`Recount`	no
`deglov`	don't print intermediate output

In the sixth and seventh columns of Table 2 are results obtained using Fiedler's method. To obtain the results cited, we first constructed the Laplacian of the graph representation of the various netlists with net weights w_{uv} equal to the inverse of the sum of each row of the adjacency matrix. In the last two columns of Table 2 are results obtained using Barnes' method.

In terms of net cut, hMetis produces the best results, followed by eig, Fiedler, and Barnes. In terms of CPU time, our technique appears overall to be the fastest, followed by hMetis and Barnes, with Fiedler a fair bit slower. The "—" in the Fiedler columns for rows `ibm15`–`ibm18` indicate that the Lanczos routine had exceeded the time-out value point of 1,800 seconds (30 minutes) to compute eigenvectors and eigenvalues.

Although for eig, the net cut is inferior to that of hMetis, we can glean significant insight into the structure of the netlist by using the placement produced by eig. We can use this information to determine whether it is better to partition the hypergraph into two or three or more blocks. For example, in the Figure 5, by using eig, we can see that the best way to partition `ibm02` is into three nearly equal-sized blocks, rather than two. eig is intended to be a complementary method to hMetis by providing insight on the number of blocks and block sizes, which hMetis cannot do in advance.

Technique eig obtains the smallest net cuts compared to other eigenvector methods because it takes both vertex and net weights into account when performing the optimization. The components of the eigenvector are more spread out than the components of the Fiedler vector or the eigenvector components used by Barnes. The plots comparing eigenvector components resulting from eig, Fiedler, and Barnes confirm that this is indeed the case. For problem `ibm01`, Fiedler's method produces a net cut nearly as small as eig: this is evidenced by the fact that the eigenvector components of Fiedler's method follow almost the same distribution as the components resulting from eig (see Figure 6).

5.2. Limits of Interchange-Based Partitioning

Multilevel partitioners use vertex-matching techniques to merge vertices of small degree into vertices of larger degree. Matchings may be selected randomly or by decreasing net size (Karypis et al. 1998). After clustering, the average vertex weight increases, but the average net degree decreases. If it is not "obvious" which nets are of largest degree, then multilevel techniques will have difficulties deciding on the right vertices to merge. The purpose of the following experiment is to see what happens when the vast majority of nets have exactly the same degree. To simplify matters, we generate block-diagonal $1{,}000 \times 1{,}000$ matrices with k orders below the diagonal and k orders above the diagonal, so most nets are connected to $2k+1$ vertices. The goal is to partition them into two blocks such that the net cut is minimized (the globally minimal net cut for such netlists is exactly $2k$).

Table 2 Bipartitioning Results for Different Eigenvector Methods

	hMetis		eig		Fiedler		Barnes	
Circuit	Net-cut	Time	Net-cut	Time	Net-cut	Time	Net-cut	Time
fract	11	0.12	11	0.05	12	0.21	28	0.07
baluP	27	0.26	29	0.15	149	1.60	228	0.22
prim1	47	0.29	88	0.12	91	0.72	243	0.21
test	50	0.42	65	0.26	205	7.53	247	0.35
struct	33	0.29	46	0.30	203	1.21	681	0.75
ind1	19	0.59	22	0.56	67	15.58	229	0.73
prim2	151	1.11	260	0.51	338	3.66	605	1.79
bio	83	0.97	143	1.62	328	8.95	660	3.85
ind2	174	3.64	386	5.80	1,576	77.09	2,057	5.01
ind3	249	4.80	397	4.64	518	64.55	1,273	8.61
avq.small	127	4.30	267	13.30	13,298	390.47	1,399	11.85
avq.large	127	4.96	270	16.27	14,869	420.38	1,142	14.76
golem3	1,339	74.29	6,160	134.52	8,611	665.82	14,808	23.75
ibm01	181	5.97	270	3.37	337	13.26	1,333	5.65
ibm02	262	13.18	772	6.59	1,077	25.87	1,510	17.06
ibm03	958	16.08	2,179	7.62	3,582	146.19	6,427	10.87
ibm04	547	17.35	781	9.63	951	218.45	5,044	15.62
ibm05	1,725	22.03	2,252	9.36	7,621	39.66	8,470	27.29
ibm06	941	28.30	1,424	14.74	5,063	77.99	11,115	19.43
ibm07	900	44.87	1,818	32.00	2,886	178.93	6,083	26.95
ibm08	1,145	63.94	2,437	30.91	3,318	389.85	14,816	44.40
ibm09	638	46.16	1,885	31.19	2,045	127.86	15,142	43.97
ibm10	1,285	82.45	3,673	122.00	5,503	817.46	12,814	70.08
ibm11	962	73.49	3,699	51.58	2,850	270.88	16,478	46.46
ibm12	1,929	101.26	7,085	54.32	7,605	877.06	16,340	67.15
ibm13	875	98.58	5,041	74.96	3,505	917.57	16,228	93.77
ibm14	1,955	235.40	3,641	160.90	4,285	1,610.72	29,034	151.73
ibm15	2,629	300.83	5,727	244.34	—	>1,800	31,925	262.66
ibm16	1,809	331.64	2,805	472.50	—	>1,800	49,683	292.85
ibm17	2,225	404.10	4,046	359.30	—	>1,800	27,288	287.86
ibm18	1,525	396.15	3,290	587.99	—	>1,800	30,702	279.64
	−46%	39%	—	—	473%	1,062%	466%	40%

In Table 3, the names of the matrices are given by circuit-k; "d.n.c." (did not converge) indicates the partitioner was not able to solve the problem.

When the density exceeds approximately 5%, the multilevel partitioner is no longer able to determine the net cut, whereas eig is able to do so even for reasonably dense problems.

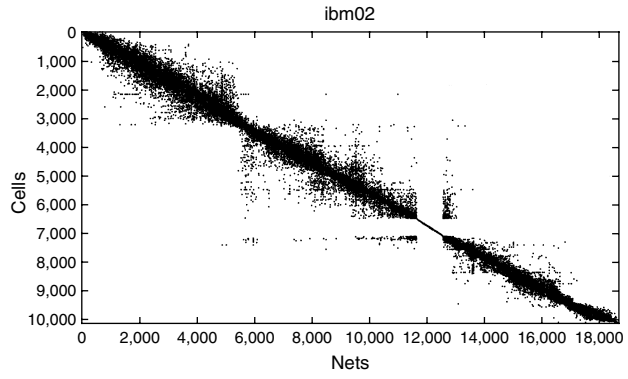

Figure 5 ibm02 **Reveals a Natural Three-Way Partition**

5.3. Bipartitioning Results with Fixed Terminals

The purpose of this experiment is to see whether there is any improvement in performance when fixed terminals are included. Typically, circuit blocks that have switching units of widely varying size and complexity (e.g., everything from the smallest inverter to the largest adder) will require more power, ground, and logistic signals from the outside, so more fixed terminals are required.

Results are provided in Table 4 where eigFT indicates eig with fixed terminal information, eig indicates no fixed terminal information is used, c is the fixed terminal scaling parameter, and μ_{ft} is the average net degree of nets that are attached to fixed terminals. The net cut in the eig column is generated by setting the fixed-terminal scaling parameter to $c = 0$; the net cut in the eigFT column is generated by finding the *best* fixed-terminal scaling parameter through trial and error.

The secondary purpose of this exercise is to gather information about the terminal-scaling parameter. The terminal-scaling parameter is related to the average size of nets that are connected to terminals:

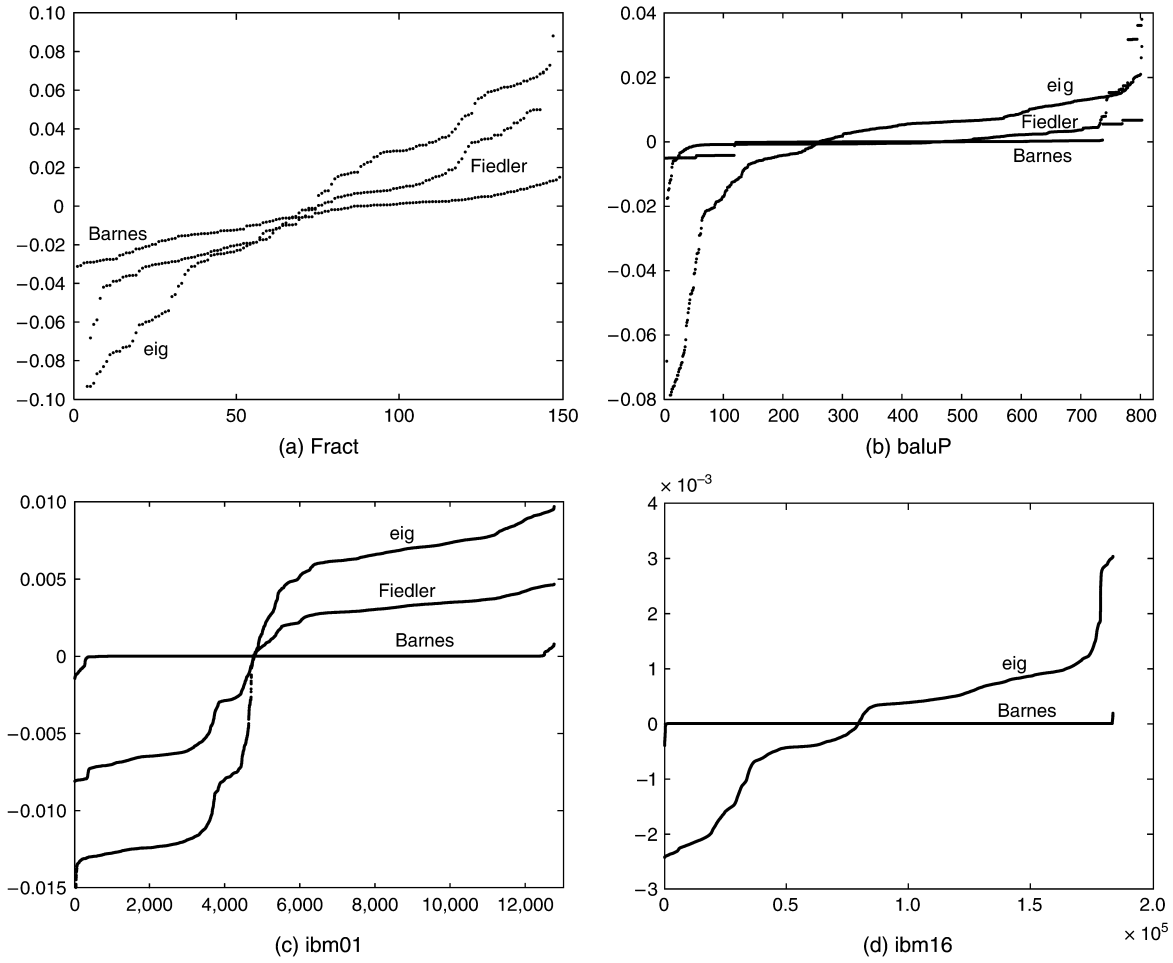

Figure 6 **Why** eig **Yields Smaller Net-Cuts than** Fiedler **and** Barnes

Note. In cases where eig obtains a much smaller net-cut than do Fiedler and Barnes (`baluP` and `ibm16`), the components of the second eigenvector obtained using eig are more spread out than eigenvector components obtained by the Fiedler and Barnes methods

witness the small values of c for test problems `test`, `struct`, `avq.small`, and `avq.large`. It is not surprising that these circuits also have higher terminal-net degree than the rest. In these circuits, the fixed terminals naturally have much impact on how vertices are ordered. Only a small amount of extra weight is required to place all fixed vertices in the correct block as in the case of `struct` (the net-cut obtained there is in fact the globally optimal net-cut with fixed termi-

Table 3 **Exploring the Limits of Interchange-Based Partitioners**

Circuit	hMetis	eig
circuit-1	2	2
circuit-10	20	20
circuit-20	40	40
circuit-23	46	46
circuit-24	48	48
circuit-25	d.n.c.	50
circuit-50	d.n.c.	100
circuit-100	d.n.c.	200
circuit-200	d.n.c.	400

Table 4 **Two-Way Net Cut Results with Fixed Terminals**

Circuit	eig	eigFT	Time	Impr. (%)	c	μ_{ft}
fract	11	11	0.05	0	0.12	4.00
prim1	88	62	0.17	−42	0.15	3.07
test	65	56	3.00	−16	0.05	9.47
struct	46	48	0.36	4	0.02	5.52
prim2	260	190	0.74	−37	0.16	1.96
bio	143	143	1.75	0	0.05	48.16
ind2	382	326	1.94	−17	0.15	6.98
ind3	397	417	6.33	5	0.10	2.10
avq.small	267	236	8.46	−13	0.05	273.72
avq.large	270	240	10.14	−12	0.05	273.76
ibm01	270	217	3.55	−24	0.15	2.12
ibm06	1,424	1,418	11.09	0	0.15	2.24
ibm09	1,885	1,887	24.80	0	0.15	2.00
ibm10	3,673	2,906	35.78	−26	0.16	2.05
ibm11	3,699	3,699	30.87	0	0.15	2.00
ibm12	7,085	5,216	32.14	−36	0.20	2.01
ibm13	4,731	2,956	41.79	−60	0.26	2.04
ibm16	2,805	2,832	245.24	1	0.15	2.14
ibm17	4,046	3,560	123.26	−14	0.21	2.00
Average improvement				−15		

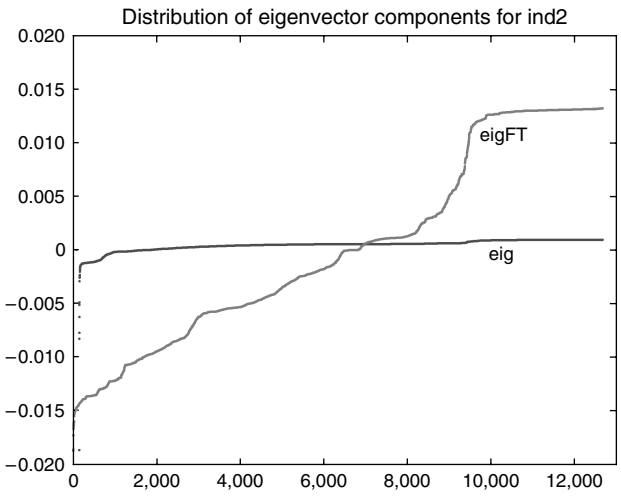

Figure 7 Fixed Terminals Cause Eigenvector Components to Follow a More Monotone Distribution

nals included). The table suggests a rule of thumb. If $\mu_{ft} < 5$, then $0.10 \leq c \leq 0.30$; if $\mu_{ft} \geq 5$, then $0 \leq c \leq 0.10$. If $c > 0.30$, then the numerical routine is dominated by the weighted (fixed-terminal) eigenvector components—in effect, it disregards placement information present in the eigenvector components from **F** so does not produce a good placement.

The results show that for the most part, including fixed terminals in the eigenvector formulation produces better vertex placements than if no fixed terminals are included. Figure 7 demonstrates that in cases where the net cut is better with fixed terminals, the eigenvector components follow a more monotone distribution.

6. Conclusions

This paper discussed a partitioning technique that uses the second eigenvector of an adjacency-like matrix. Our technique yields superior results to two other well-known eigenvector techniques because we assign weights to vertices, nets, and fixed vertices. We pointed out that by weighting vertices and nets, the components of the eigenvector that are used to determine placements follow a more monotone distribution, which ultimately results in better-quality partitions. We also extended the technique to incorporate fixed vertices that represent input/output terminals of a chip, and observed that the monotonicity of the eigenvector components is further strengthened and in many cases, the net cut improved. Since our technique uses the second largest eigenvector of the modified adjacency matrix, we were able to solve the largest test problems in reasonable time. In particular, our technique scales approximately linearly with the number of vertices. An important feature of this new eigenvector model is that it complements the multilevel partitioners like hMetis by providing important insight into the "naturally occurring" number of blocks and block sizes that are not easily gauged in these competitive partitioners.

Acknowledgments

The first author is currently with the IBM Thomas J. Watson Research Center, 1101 Kitchawan Road, Yorktown Heights, NY 10598. The second author was supported by a from the National Sciences and Engineering Research Council of Canada. The authors would also like to thank Miguel Anjos and Andrew Kennings for helpful suggestions.

References

Alpert, C. J. 1998. The ISPD98 circuit benchmark suite. *Proc. 1998 Internat. Sympos. Physical Design.* ACM Press, New York, 80–85.

Alpert, C. J., A. B. Kahng, S. Z. Yao. 1999. Spectral partitioning with multiple eigenvectors. *Discrete Appl. Math.* **90** 3–26.

Alpert, C. J., A. E. Caldwell, A. B. Kahng, I. Markov. 2000. Hypergraph partitioning with fixed vertices. *IEEE Trans. Comput.-Aided Design Circuits Systems* **19** 267–272.

Barnes, E. R. 1982. An algorithm for partitioning the nodes of a graph. *SIAM J. Algebraic Discrete Methods* **3** 541–550.

Caldwell, A., I. Markov, A. B. Kahng. 1999. The ISPD99 benchmark suite.

Chan, P. K., M. Schlag, J. Zien. 1994. Spectral K-way ratio-cut partitioning. *IEEE Trans. Comput.-Aided Design Integrated Circuits Systems* **13** 1088–1096.

Demmel, J. W. 1997. *Applied Numerical Linear Algebra*. Society of Industrial and Applied Mathematics Press, Philadelphia, PA.

Fiedler, M. 1973. Algebraic connectivity of graphs. *Czechoslovak Math. J.* **23** 298–305.

Frankle, J., R. M. Karp. 1986. Circuit placements and cost bounds by eigenvector decomposition. *Proc. 1986 Internat. Conf. Comput.-Aided Design, San Jose, CA.* ACM Press, New York, 414–417.

Hall, K. M. 1970. An *r*-dimensional quadratic placement algorithm. *Management Sci.* **17** 219–229.

Hendrickson, B., R. Leland, R. van Driessche. 1996. Enhancing data locality by using terminal propagation. *Proc. 29th Hawaii Internat. Conf. System Sci.*, 565–574.

Ihler, E., D. Wagner, F. Wagner. 1993. Modelling hypergraphs by graphs with the same mincut properties. *Inform. Processing Lett.* **45** 171–175.

Karypis, G., V. Kumar. 1998. hMETIS: A hypergraph partitioning package, version 1.5.3. Department of Computer Science/Army HPC Research Center, University of Minnesota, Minneapolis, MN.

Karypis, G., R. Aggarwal, V. Kumar, S. Shekhar. 1997. Multilevel hypergraph partitioning: Application in VLSI domain. *Proc. 34th ACM/IEEE Design Automation Conf.* ACM Press, New York, 526–529.

Kozminski, K. 1991. Benchmarks for layout synthesis-evolution and current status. *Proc. 28th IEEE/ACM Design Automation Conf.* ACM Press, New York, 265–270.

Otten, R. H. J. M. 1982. Eigensolutions in top-down layout design *Proc. Internat. Sympos. Circuits Systems*, IEEE Press, Piscataway, NJ, 1017–1020.

Pothen, A., H. D. Simon, K. P. Liou. 1990. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.* **11** 430–452.

Riess, B. M., K. Doll, F. M. Johannes. 1994. Partitioning very large circuits using analytical placement techniques. *Proc. ACM Design Automation Conf.* ACM Press, New York, 646–651.

Tsay, R.-S., E. S. Kuh. 1991. A unified approach to partitioning and placement. *IEEE Trans. Circuits Systems* **38** 521–533.

Watkins, D. S. 1991. *Fundamentals of Matrix Computations*. Wiley, New York.