Hybrid Spectral/Iterative Partitioning

Jason Y. Zien
IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

Pak K. Chan, Martine Schlag
Computer Engineering, UCSC
Santa Cruz, CA 95064

Abstract
We develop a new multi-way, hybrid spectral/iterative hypergraph partitioning algorithm that combines the strengths of spectral partitioners and iterative improvement algorithms to create a new class of partitioners. We use spectral information (the eigenvectors of a graph) to generate initial partitions, influence the selection of iterative improvement moves, and break out of local minima. Our 3-way and 4-way partitioning results exhibit significant improvement over current published results, demonstrating the effectiveness of our new method. Our hybrid algorithm produces an improvement of 25% over GFM for 3-way partitions, 41% improvement over GFM for 4-way partitions, and 58% improvement over MLF for 4-way partitions.

1 Introduction
This paper describes a new multi-way, hybrid spectral/iterative, hypergraph partitioning algorithm, which consolidates iterative and spectral information in a multi-level partitioning framework. Previously, researchers combined the two approaches by finding a solution using a spectral algorithm and then performing iterative improvement on it [1, 2, 3]. The simultaneous use of spectral and traditional gain costs is found in the single-pass constructive heuristic presented in [4]; but it is not used for iterative improvement. In contrast, our work provides a tightly coupled integration of spectral and iterative improvement methods. Our contributions include the use of circular orderings to generate multiple initial partitions, using spectral information within a Kernighan-Lin/Fiduccia-Mattheyses iterative improvement algorithm [5] to break ties in gain, and using spectral information to break out of local minima which may trap standard iterative improvement algorithms. Our partitioner produces significant improvements over [6] and [7].

1.1 Spectral Partitioning and Maximum Sum Vector Partitioning
Given a graph with \( n \) vertices, we wish to find \( k \) partitions of this graph. An \( n \times n \) adjacency matrix, \( A \), has entries \( a_{ij} \) that represent the weight of an edge between vertices \( v_i \) and \( v_j \). The \( n \times n \) diagonal degree matrix, \( D \), has entries \( d_{ii} \) equal to the sum of the weights of all edges on vertex \( v_i \). \( M \) is the \( n \times n \) diagonal matrix whose \( m_{ii} \) entry is the size of vertex \( i \). The Laplacian matrix is \( Q = D - A \). Spectral partitioning uses the eigenvectors associated with the smallest eigenvalues of \( Q \). The eigenvectors are the solution to a relaxed version of the partitioning problem known as the quadratic placement problem. A generalization of the spectral partitioning problem which takes vertex sizes \( m_{ii} \) into account was solved by [3]. Different numbers of eigenvectors can be used in creating partitions. For instance, the authors of [8] use the \( 2^{nd} \) eigenvector to form partitions; multiple eigenvectors are used in [1, 4, 9].

In general, let us assume we are using \( d \) eigenvectors. Let \( X_d \) be the \( n \times d \) matrix composed of the first \( d \) eigenvectors of \( Q \). Let \( \Lambda_d \) be the \( d \times d \) diagonal matrix composed of the first \( d \) eigenvalues \( (\lambda_1, ..., \lambda_d) \) of \( Q \). \( H_d \) is a diagonal matrix whose entries are all \( \gamma \), with \( \gamma \geq \lambda_1 \). In maximum sum vector partitioning (MSVP), the scaled matrix of eigenvectors is represented by \( V_d = MX_d \sqrt{\gamma H_d} - \Lambda_d \) where \( X_d \) and \( \Lambda_d \) satisfy \( QX_d = MX_d\Lambda_d \) [10], with \( \gamma = |\lambda_1| + |\lambda_2| \). This is an extension of the methods of [9, 11]. An experimental verification of this embedding choice is given in [10].

Let the vector \( \nu_i \) be the \( j \)th row of \( V_d \). In MSVP we wish to divide the vectors \( \nu_i \) into \( k \) distinct sets of vectors, corresponding to the \( k \) partitions. Let \( T_h = \sum_{i \in P_h} \nu_i \) be sum of each of those sets of vectors with \( 1 \leq h \leq k \). The goal of MSVP is to maximize \( \sum_{h=1}^{k} \| T_h \|_2^2 \). When \( d = n \), it has been shown that maximum sum vector partitioning is equivalent to minimizing the sum of the edges cut in graph partitioning [9]. More eigenvectors provide a closer approximation to the graph partitioning problem. However, there is a dilemma as a partitioner uses more eigenvectors: the multi-dimensional embedding becomes increasingly difficult to take advantage of. Note that when \( d = n \), the eigenvector computation can be omitted and the maximum sum partitioning methods can be applied to the graph matrix.

2 Hybrid Spectral/Iterative Partitioning
The classic Kernighan-Lin and Fiduccia-Mattheyses (KLFM) algorithm is the basis for most modern iterative partitioning algorithms. Because KLFM algorithms are so sensitive to initial starting points, a large number of random starts is required to obtain good solutions. Researchers have sought ways to create more stable performance by using clustering
and multiple levels of hierarchy.

The primary advantage of spectral algorithms is that they are able to find a globally optimal solution to a relaxed version of the partitioning problem. They have been found to perform well on partitioning problems using the ratio-cut cost function [8], although the relaxed continuous-space solution may not be close to the optimal discrete partitioning solution. However, spectral partitioning methods have their weaknesses. For example, constraints such as partition size, partition topology, and pin limits are difficult to incorporate, although there are some exceptions [4].

Furthermore, due to the conversion from a hypergraph into a graph, optimal solutions for the resulting graph do not necessarily correspond to optimal solutions for the original hypergraph, although they are close. Since only one deterministic solution is available, there are no alternatives if the solution is poor. With all of the above difficulties, it is no wonder that spectral algorithms have been found to be worse than some iterative partitioning algorithms [12]. Researchers have also tried to combine spectral and iterative methods by using a spectral partitioner to obtain the initial partition for an iterative method, but the results were inferior to using random starts [2]. What if we consolidate spectral information with iterative improvement algorithms to obtain one unified method which combines the advantages of both methods? We can utilize the global information from a spectral algorithm within an iterative partitioning framework and gain the benefits of both methods.

2.1 Overview

Figure 1 illustrates our multi-level partitioner. Spectral information can be used in the contraction algorithms, initial partitioning, and in the iterative improvement process. The eigenvectors and eigenvalues of a graph are computed during the initial partitioning. We can optionally recompute the eigenvectors in the expansion phase if desired. In our benchmarks, we recompute eigenvectors during expansion, unless otherwise noted.

Our iterative improvement algorithm is a KLFM style algorithm. We have extended it to support $k$-way partitions by iteratively working on one partition at a time. Each partition is used in turn as the target partition. The only moves considered are those in which vertices move from one of the $k - 1$ other partitions into the target partition, or out of the target partition and into one of the $k - 1$ other partitions. We use spectral information in the updating of the neighbor gains to break ties and influence the move sequence. Our implementation uses heaps to store vertex gains rather than buckets, in order to support the non-integer gains in our hybrid spectral/iterative algorithm.

2.2 Using Spectral Information

There are 3 key areas in iterative improvement algorithms that can be augmented using spectral information: 1. initial partition generation, 2. breaking ties when choosing the next move to make, and 3. jumping out of local minima.

---

**ALGORITHM**: MLH  
**INPUT**: hypergraph, target number of partitions $k$, maximum contracted size  
**OUTPUT**: $k$-way partitioning solution

**METHOD**:  
Read hypergraph  
While (graph > maximum contracted size) {  
  Contract graph  
}  
for $i = 1$ to max.iterations {  
  Create Initial $k$-way Partitioning  
  Rotary KLFM or Flow Iterative Improvement  
}  
Expand graph, map solution  
Rotary KLFM or Flow Iterative Improvement

Figure 1: The multi-level partitioning algorithm.

2.2.1 Generating Initial Partition

We focus on solving the $k$-way partitioning problem for small values of $k$, such as 2, 3 or 4. Higher dimensional techniques are discussed in [13, 14]. We use the 2nd and 3rd scaled eigenvectors to obtain a two-dimensional embedding. Henceforth, we refer to scaled eigenvectors, $V_d$, simply as eigenvectors. We limit ourselves to planar embeddings for practical reasons: it is easy to generate orderings of vertices, the computation of the first 3 eigenvectors is fast, and we can visually examine and understand our results.

As discussed earlier, the partitioning result based on the eigenvectors is remote from the original problem. Figure 2 illustrates the “loose” correspondence between the initial and final solution costs for our SWEEP algorithm on 100 initial starting points. Because iterative improvement itself is sensitive to the starting solution, we generate many initial partitions.

---

**Figure 2**: Initial (top) and final (bottom) hyperedge cuts for circuit p2.ga.
We use soft constraints to enforce balance in partitions, as follows: \( \text{bal}_l \leq (1 + \text{bal})||P_l|| \leq (1 + \text{bal})||P_h|| \). Here, \( P_h \) is an arbitrary partition, \( h \), and \( ||P_l|| \) is the target partition size, which is the sum of the sizes of all vertices divided by \( k \), the number of partitions desired. By default, \( \text{bal} = 0.1 \).

After each vertex move in iterative improvement, the neighbors of the moved vertex must have their gains updated. The \( \text{cut gain} \) is the weight of the hyperedges that become uncut by a move.

The \textit{spectral gain} comes from a 2-dimensional placement of vertices in the plane. Recall that each vector in the plane represents a vertex. Let \( T_h \) be the sum of all the vectors in partition \( P_h \). The spectral gain of a vector \( \nu_i \) with respect to a destination partition \( P_h \) is calculated as \( ||\nu_i + T_h|| \). The vector sums, \( T_h \), of each of the \( k \) target partitions are calculated at the beginning of each pass, and are not dynamically updated as moves are made. The spectral gain is negated when the target destination is not the same partition as the last moved vertex. This encourages groups of vertices to move to the same target destination in connected groups, as in the FM-CLIP algorithm [12].

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

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

### 3 Multi-level Spectral Partitioning

In this section, we consider a number of implementation choices that would have a significant effect on the final partitioning result.

**Hypergraph to graph conversion:** We use the star graph model to convert hypergraphs into graphs, as in [4]. By following the method of [11], we create the edge weights of the star graph to be \( \sqrt{d} \) to minimize the discrepancy between the best and worst case quadratic placement of nodes on a unit span ([10] gives details).

**Contraction algorithm:** We use the heavy edge matching cost function due to [15]. This cost function contracts the edges with the highest weights, which would be the most undesirable to cut. We convert the hypergraph into a graph, sort the edges based on the edge weight, and match any unmatched pairs of vertices as edges are picked in order of decreasing cost.

**Levels of hierarchy:** One of the contributing parameters of contraction is the clustering or matching method, which has been studied extensively. Less attention has been paid to the levels of contraction. As the number of levels of hierarchy increases, the initial problem solution becomes more remote from the original problem. The best solutions may be precluded due to the contraction algorithm. This is offset by the
The fact that more levels of iterative improvement can be run when there are more hierarchical levels.

Many researchers have used a fixed number of levels of contraction. Other researchers have chosen to contract the graph until the number of nodes reaches a target size, such as 35 [7], or 100 nodes [15]. We ran some benchmark circuits for the unit-size vertex tests, and evaluated the results for different numbers of max nodes, and also an unlimited number of nodes (which corresponds to no levels of contraction). Our evaluation suggested that we should use a 200-node maximum contracted graph size for 2-way partitioning, and 400 nodes for 3- and 4-way partitioning.

Disconnected graphs: Disconnected graphs lead to extremely unbalanced partitions (partitions where each connected component is in its own partition). We handle this problem by introducing dummy edges to connect the graph. These edges are introduced immediately before the eigensolver is invoked, and are then removed immediately after, so that they do not affect any edge cut costs. For star graphs, we introduce a hyperedge which is connected to an arbitrary vertex in each of the components. Others have used higher-order eigenvectors to handle disconnected graphs [9], Shau [16] used different pairs of lower-order eigenvectors. Since we did not observe unusually poor performance for graphs using our “dummy-edge” method, we did not resort to higher order eigenvectors.

4 Results on MCNC Benchmarks

We compared our hybrid algorithms against existing published results. Our partitioner was implemented in C++. We replaced the traditional KLFM bucket implementation with heaps in order to support non-integer gains due to our hybrid cost function, Equation (1). We contracted graphs until there were at most 400 nodes, and generated 20 initial partitions (p = 20) using the SWEEBP initial partitioning method. We examined both the HYBRID tie-breaking algorithm as well as the HYBRID Alt algorithm that alternates between spectral and iterative gain costs.

Our partitioner performs well on 2-way partitioning, but is not superior to the best 2-way partitioners. The detailed benchmark results are in [10]. The strength of our partitioner lies in combining global spectral information with iterative improvement. Global information becomes even more important in 3-way and 4-way partitioning, which is perhaps why our 3-way and 4-way results, are substantially better than other partitioners.

4.1 Comparison against GFM and PD

We compared the number of hyperedges cut by our hybrid algorithms to the GFM [6] and Primal-Dual (PD) algorithm [17], using the actual vertex sizes. We omitted graph t5 because in 4-way partitioning, the largest node in t5 is larger than that allowed by balance constraints. Each partition was allowed a balance factor of $bal = 0.5$.

Tables 1 and 2 show the results of our hybrid partitioner compared with the previously published results of [6]. The last row, Gmean, is the geometric mean over all of the circuits. In the three-way partitioning, our HYBRID ALT method gives a 19% improvement, and HYBRID gives a 25% improvement over GFM. In four-way partitioning, HYBRID ALT is 41% better than GFM and HYBRID is 38% better than GFM.

As a control experiment, we also show the results of our implementation of a multi-level, k-way FM-CLIP algorithm to determine whether the hybrid algorithm was really the source of improvement, rather than other factors, such as contraction method, levels of contraction, or our multi-way improvement method. We found that our best hybrid method was 7% better than FM-CLIP in 3-way partitioning, and 19% better in 4-way partitionings. As the number of desired partitions increases, use of global spectral information becomes more important. Although multi-level contraction substantially improves upon the standard KLFM cut gain, by incorporating spectral costs as well, even further improvements can be made.

4.2 Comparison against MLF and GORDIAN

We compared our hybrid algorithm to the MLF [7] and GORDIAN algorithms [18]. We also report our multi-level implementation of the FM-CLIP algorithm. These tests use unit vertex sizes with a balance factor of $bal = 0.1$. Table 3 shows that our HYBRID partitioner is 12.1% better than our implementation of the multi-level FM-CLIP algorithm, 58.4% better than the MLF algorithm, and 175.7% better than GORDIAN in terms of the number of hyperedges cut. Our hybrid method is able to effectively use the global spectral information to substantially improve upon other partitioning methods.

The execution time of our partitioner is slower than

<table>
<thead>
<tr>
<th>Circuit</th>
<th>PD</th>
<th>GFM</th>
<th>KLFM</th>
<th>HybridA</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1.gas</td>
<td>66</td>
<td>74</td>
<td>66</td>
<td>66</td>
<td>68</td>
</tr>
<tr>
<td>p1.sc</td>
<td>77</td>
<td>76</td>
<td>74</td>
<td>74</td>
<td>98</td>
</tr>
<tr>
<td>p2.gas</td>
<td>377</td>
<td>235</td>
<td>178</td>
<td>178</td>
<td>178</td>
</tr>
<tr>
<td>p2.sc</td>
<td>297</td>
<td>174</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>12</td>
<td>81</td>
<td>81</td>
<td>81</td>
<td>81</td>
<td>33</td>
</tr>
<tr>
<td>13</td>
<td>168</td>
<td>101</td>
<td>81</td>
<td>77</td>
<td>33</td>
</tr>
<tr>
<td>15</td>
<td>80</td>
<td>85</td>
<td>88</td>
<td>85</td>
<td>83</td>
</tr>
<tr>
<td>16</td>
<td>117</td>
<td>124</td>
<td>94</td>
<td>94</td>
<td>94</td>
</tr>
<tr>
<td>Genmed</td>
<td>125.9</td>
<td>113.4</td>
<td>94.53</td>
<td>94.17</td>
<td>88.76</td>
</tr>
</tbody>
</table>

Table 1: Comparison of hyperedges cut of our Hybrid and HybridA 3-way partitioning against the GFM and PD partitioner using actual vertex sizes.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>PD</th>
<th>GFM</th>
<th>KLFM</th>
<th>HybridA</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1.gas</td>
<td>102</td>
<td>107</td>
<td>98</td>
<td>87</td>
<td>91</td>
</tr>
<tr>
<td>p1.sc</td>
<td>107</td>
<td>110</td>
<td>90</td>
<td>86</td>
<td>89</td>
</tr>
<tr>
<td>p2.gas</td>
<td>447</td>
<td>335</td>
<td>223</td>
<td>223</td>
<td>223</td>
</tr>
<tr>
<td>p2.sc</td>
<td>447</td>
<td>335</td>
<td>223</td>
<td>223</td>
<td>223</td>
</tr>
<tr>
<td>12</td>
<td>217</td>
<td>185</td>
<td>131</td>
<td>135</td>
<td>135</td>
</tr>
<tr>
<td>13</td>
<td>170</td>
<td>152</td>
<td>104</td>
<td>104</td>
<td>104</td>
</tr>
<tr>
<td>14</td>
<td>213</td>
<td>198</td>
<td>144</td>
<td>144</td>
<td>144</td>
</tr>
<tr>
<td>16</td>
<td>186</td>
<td>144</td>
<td>108</td>
<td>108</td>
<td>108</td>
</tr>
<tr>
<td>Genmed</td>
<td>216.4</td>
<td>182.7</td>
<td>132.49</td>
<td>132.88</td>
<td>124.34</td>
</tr>
</tbody>
</table>

Table 2: Comparison of hyperedges cut of our Hybrid and HybridA 4-way partitioning against the GFM and PD partitioner using actual vertex sizes.
Table 3: Comparison of hyperedges cut of our Hybrid and HybridA 4-way partitioning against MLF and GORDIAN using unit-size vertices.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Hybrid</th>
<th>HybridA</th>
<th>KFMC</th>
<th>MLF</th>
<th>GORDIAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>honored</td>
<td>238</td>
<td>210</td>
<td>280</td>
<td>311</td>
<td>479</td>
</tr>
<tr>
<td>industry2</td>
<td>339</td>
<td>388</td>
<td>458</td>
<td>368</td>
<td>1172</td>
</tr>
<tr>
<td>industry5</td>
<td>163</td>
<td>172</td>
<td>184</td>
<td>650</td>
<td>1175</td>
</tr>
<tr>
<td>94.5gs</td>
<td>24</td>
<td>76</td>
<td>112</td>
<td>72</td>
<td>138</td>
</tr>
<tr>
<td>s138.07</td>
<td>136</td>
<td>113</td>
<td>144</td>
<td>472</td>
<td>550</td>
</tr>
<tr>
<td>s156.50</td>
<td>132</td>
<td>119</td>
<td>147</td>
<td>347</td>
<td>878</td>
</tr>
<tr>
<td>German</td>
<td>229.51</td>
<td>211.40</td>
<td>335.64</td>
<td>306.92</td>
<td>113.45</td>
</tr>
</tbody>
</table>

that of other partitioners due to the use of heap data structures. Heaps are necessary in order to support non-integer gains in iterative improvement for our hybrid algorithm. Three-way partitioning execution time results vary from 80 seconds to 1100 seconds on a Sun Ultra 170.

5 Conclusion

We have developed a new hybrid spectral/iterative partitioning algorithm and have demonstrated that it performs better than the best known 3 and 4-way partitioners. Our hybrid algorithm is only the first step in developing newer, more sophisticated iterative improvement algorithms. The key ideas that need further research includes the use of new gain cost functions to influence move selection and new objective functions that allow iterative improvement algorithms to break out of local minima. Future research includes using higher dimensions (more eigenvectors) to generate the initial partitions, and investigating methods for speeding up hybrid partitioning.

Acknowledgments

The authors acknowledge the support from the National Science Foundation under Grant number MIP-9223740.

References


