Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 99-034

Multilevel Algorithms for Multi-Constraint Hypergraph Partitioning

George Karypis

November 11, 1999

Multilevel Algorithms for Multi-Constraint Hypergraph Partitioning*

George Karypis

University of Minnesota, Department of Computer Science / Army HPC Research Center
Minneapolis, MN 55455
Technical Report #99–034

karypis@cs.umn.edu

Last updated on November 11, 1999 at 7:06am

Abstract

Traditional hypergraph partitioning algorithms compute a bisection a graph such that the number of hyperedges that are cut by the partitioning is minimized and each partition has an equal number of vertices. The task of minimizing the cut can be considered as the ***objective*** and the requirement that the partitions will be of the same size can be considered as the ***constraint***. In this paper we extend the partitioning problem by incorporating an arbitrary number of balancing constraints. In our formulation, a vector of weights is assigned to each vertex, and the goal is to produce a bisection such that the partitioning satisfies a balancing constraint associated with each weight, while attempting to minimize the cut. We present new multi-constraint hypergraph partitioning algorithms that are based on the multilevel partitioning paradigm. We experimentally evaluate the effectiveness of our multi-constraint partitioners on a variety of synthetically generated problems.

*This work was supported by NSF CCR-9972519, by Army Research Office contract DA/DAAG55-98-1-0441, by the DOE ASCI program, and by Army High Performance Computing Research Center contract number DAAH04-95-C-0008. Access to computing facilities was provided by AHPCRC, Minnesota Supercomputer Institute. Related papers are available via WWW at URL: **http://www.cs.umn.edu/~karypis**

1 Introduction

The traditional hypergraph bisection problem divides the vertices into two partitions such that the hyperedge-cut is minimized and each partition has roughly an equal number of vertices (or in the case of weighted hypergraphs, the sum of the vertex-weights in each partition is the same). The task of minimizing the cut can be considered as the ***objective*** and the requirement that the partitions are of the same size can be considered as the ***constraint***. This single-objective single-constraint hypergraph partitioning formulation has extensive application to many areas, including VLSI design [3], efficient storage of large databases on disks [11], and data mining [10]. Unfortunately, this problem formulation is not sufficient to model the underlying partitioning requirements of most problems arising in VLSI design, as most of the problems are inherently multi-objective and multi-constraint.

In this paper we present a generalized hypergraph bisection problem in which a vector of weights is assigned to each vertex. The goal is to produce a bisection of the hypergraph such that it satisfies a balancing constraint associated with each one of the weights, while attempting to minimize the cut (*i.e.*, the objective function). We refer to it as a ***multi-constraint*** hypergraph partitioning problem. This multi-constraint framework can be used to compute partitionings for a number of interesting problems. For instance, using this framework we can compute circuit partitionings that not only minimize the number of nets being cut, but also simultaneously balance the area, power, noise, nets, pins, *etc.*, of the partitions. Such partitionings have the potential of leading to better, more reliable, predictable, and robust VLSI design methodologies. This formulation builds upon the recently developed multi-constraint formulations for the graph partitioning problem [6] that has been shown to have extensive applications in load balancing multi-phase and multi-physics numerical simulations on parallel computers.

We present new multi-constraint hypergraph partitioning algorithms that are based on the multilevel hypergraph partitioning paradigm [8, 2, 12]. Our work focuses on developing new types of heuristics for coarsening, and refinement that are capable of successfully handling multiple constraints. We experimentally evaluate the effectiveness of our multi-constraint partitioner on a variety of synthetically generated problems derived from the ISPD98 benchmark suite [1]. Our experiments show that our multilevel multi-constraint hypergraph partitioning algorithms are able to produce high quality partitionings that satisfy the multiple balancing constraints, in a relatively small amount of time. Comparing the quality of these multi-constraint partitionings to those of the (much easier) single-constraint partitionings, we see that our algorithms lead to a moderate increase in the number of hyperedges that are cut by the partitioning.

2 Multi-Constraint Bisection Definition

Consider a hypergraph $G = (V, E)$, such that each vertex $v \in V$ has a weight vector $\boldsymbol{w}^v$ of size m associated with it, and each hyperedge $e \in E$ has a scalar weight w^e. Let $[l_i, u_i]$ for $i = 1, 2, \ldots, m$, be m intervals such that $l_i < u_i$ and $l_i + u_i = 1$. Let P be a vector of size $|V|$, such that for each vertex v, $P[v]$ is either one or two, depending on which partition v belongs to, *i.e.*, P is the bisection vector.

We place no restrictions on the weights of the hyperedges but we will assume, without loss of generality, that the weight vectors of the vertices satisfy the property that $\sum_{\forall v \in V} w_i^v = 1.0$ for $i = 1, 2, \ldots, m$. If the vertex weights do not satisfy the above property, we can divide each w_i^v by $\sum_{\forall v \in V} w_i^v$ to ensure that the property is satisfied. Note that this normalization does not in any way limit our modeling ability.

We define the multi-constraint hypergraph bisection problem as follows: Compute a bisection P of V that minimizes the hyperedge cut and at the same time, the following set of constraints is satisfied:

$$l_i \leq \sum_{\forall v \in V : P[v]=1} w_i^v \leq u_i \text{ and } l_i \leq \sum_{\forall v \in V : P[v]=2} w_i^v \leq u_i \text{ for } i = 1, 2, \ldots, m. \quad (1)$$

Definitions and Notations In this section we introduce some definitions and notations that are used through-out the paper.

1. Given a hypergraph $G = (V, E)$, a vector of m weights associated with each vertex, a bisection vector P, and a set of m balance-tolerance intervals $[l_i, u_i]$, we say that P is a ***feasible*** solution for the bisection problem if

Equation 1 is satisfied. That is, a bisection is feasible if it satisfies all the balancing constraints.

2. For each vertex v, we define ***gain*** to be the reduction in the value of the objective function (*e.g.*, cut) achieved by moving v from the partition that it belongs to to the other partition.

3. Given a set of objects A such that each object $x \in A$ has a weight-vector $\boldsymbol{w}^x$ of size m associated with it, we define w_i^A to be the sum of the ith weights of the objects in the set; *i.e.*, $w_i^A = \sum_{\forall x \in A} w_i^x$.

3 Multilevel Algorithm for Multi-Constraint Partitioning

During the last few years, hypergraph partitioning algorithms based on the multilevel paradigm have gained widespread acceptance as they provide extremely high quality partitionings, they are very fast, and they can scale to hypergraphs containing several hundred thousands of vertices [8, 2, 12].

Multilevel partitioning algorithms consist of three phases: (i) coarsening phase, (ii) initial partitioning phase, and (iii) uncoarsening (or refinement) phase. During the coarsening phase, a sequence of successively coarser hypergraphs is constructed from the original hypergraph such that the number of vertices in successive coarser hypergraphs is smaller. In the initial partitioning phase, a partitioning of the coarsest hypergraph is computed, using a conventional partitioning algorithm. Finally, during the uncoarsening phase, starting with the coarsest hypergraph, the partitioning of the hypergraph is successively projected to the next level finer hypergraph, and refined using a local partitioning refinement heuristic.

In the rest of this section, we present a multilevel recursive bisection algorithm for solving the multi-constraint partitioning problem. In particular, we present algorithms for the three phases of the multilevel bisection algorithm, namely coarsening, initial bisection, and bisection refinement during the uncoarsening phase.

3.1 Coarsening Phase

During the coarsening phase, a sequence of successively smaller hypergraphs is constructed by finding groups of vertices and merging them together to form the vertices of the next level coarser hypergraph. A number of schemes have been developed for selecting what groups of vertices will be merged together to form single vertices in the next level coarse hypergraphs [7, 8, 2, 12]. Of these schemes, the ***first-choice*** (FC) scheme [7], has been experimentally shown to produce high quality bisections.

The easiest way to understand the FC scheme is to think of the graph representation of the hypergraph, in which each hyperedge e is replaced by a clique [9] in which each edge has a weight of $w/(|e|-1)$, where w and $|e|$ are the weight and the size of the original hyperedge, respectively. In the FC scheme [7], the vertices are visited in a random order, and for each vertex v, the edge incident on v with the highest edge-weight is marked. Once all the vertices have been visited, the unmarked edges are removed, and each one of the connected components of the resulting graph becomes a set of vertices to be merged together.

The FC scheme tends to remove a large amount of the exposed hyperedge-weight in successive coarse hypergraphs, and thus makes it easy to find high quality initial bisections that require little refinement during the uncoarsening phase. In the context of multi-constraint partitioning, this feature of the FC scheme is equally applicable, and is useful for constructing successive coarse hypergraphs. However, one can also use the coarsening process to try to reduce the inherently difficulty of the load balancing problem due to the presence of multiple weights. In general, it is easier to compute a balanced bisection if the values of the different elements of every weight vector are not significantly different. In the simplest case, if for every vertex v, $w_1^v = w_2^v = \cdots = w_m^v$, then the m-weight balancing problem becomes identical to that of balancing a single weight. So during coarsening, one should try (whenever possible) to collapse groups of vertices so as to minimize the differences among the weights of the merged vertex.

We modified the FC scheme to use such a *balancing* principle in the following way. Again, we visit the vertices in a random order, but now instead of marking the edges with the heaviest weight, we first select a set of edges whose weight is no more than 10% lower than the weight of the heaviest-weight edge, and then among them we mark the one that will lead to the most balanced merged vertex, as defined by the ratio of the maximum weight over the average weight of the resulting weight vector. We will refer to this scheme as the *balanced first choice* scheme (BFC).

3.2 Initial Partitioning Phase

The goal of the initial partitioning phase of a multilevel algorithm is to compute a bisection of the coarsest hypergraph such that the balancing constraint is satisfied and the partitioning objective is optimized. In our multi-constraint partitioning algorithm, the bisection of the coarsest hypergraph is computed by performing multiple random bisections followed by a multi-constraint FM refinement step (described in the next section) to improve the quality of the bisections. In particular, we perform ten different random bisections, and select the one with the smallest hyperedge cut as the final solution. This approach is similar to that used in single-constraint multilevel hypergraph partitioning [8, 2].

3.3 Uncoarsening Phase

During the uncoarsening phase, a partitioning of the coarser hypergraph is successively projected to the next level finer hypergraph, and a partitioning refinement algorithm is used to optimize the objective function without violating the balancing constraints.

A class of local refinement algorithms that tend to produce very good results when the vertices have a single weight [8], are those that are based on the FM algorithm [4]. The FM algorithm starts by inserting all the vertices into two max-priority queues, one for each partition, according to their gains. Initially all vertices are *unlocked*, *i.e.*, they are free to move to the other partition. The algorithm iteratively selects an unlocked vertex v from the top of the priority queue from one of the partitions (source partition) and moves it to the other partition (target partition). The source partition is determined based on whether the current bisection is a feasible solution or not. If it is feasible, then the partition that contains the highest gain vertex becomes the source. On the other hand, if it is not feasible (*i.e.*, the balancing constraint is violated), the partition that contains the largest number of vertices, becomes the source. When a vertex v is moved, it is *locked* and the gain of the vertices adjacent to v are updated. After each vertex movement, the algorithm records the value of the objective function achieved at this point and whether or not the current bisection if feasible or not. A single pass of the algorithm ends when there are no more unlocked vertices. Then, the recorded values of the objective function are checked, and the point where the minimum value was achieved while achieving a feasible solution, is selected, and all vertices that were moved after that point are moved back to their original partition. Now, this becomes the initial partitioning for the next pass of the algorithm.

This single constraint FM refinement algorithm can be directly extended when the vertices have multiple weights by modifying the source-partition selection scheme. In this modified algorithm, the source partition is selected as follows. If the current bisection is feasible, then similarly to the single-weight FM algorithm, the partition that contains the highest gain vertex is selected to be the source. On the other hand, if the current bisection is infeasible, then the source partition is determined based on which partition is the largest. However, unlike the single-weight bisection problem, in the case of multiple weights, we may have both partitions being "overweight", for different weights. For example for a two-weight problem, we may have that the first partition contains more than the required total weight with respect to the first weight, whereas the second partition contains more with respect to the second weight. In our algorithm the source partition is the one that contains the most weight with respect to any single weight. For example, in the case of a two-weight problem and a 45-55 balancing constraint for each one of the weights, if (.56, .40) and (.44, .60) are the fractions of the two weights for partitions A and B, respectively, then our algorithm will select B to be the source, as .60 is greater than .56 (that A contains with respect to the first weight). We will refer to this algorithm as FM1.

One of the problems of FM1 is that it may make a large number of moves before it can reach to a feasible solution, or in the worst case fail to reach it all together. This is because it selects the highest gain vertex, irrespective of the relative weights of this vertex. For instance, in the previous example, we selected to move a vertex from B, so that we can reduce w_2^B. However, the highest gain vertex v from B, may have a weight vector such that w_2^v is much smaller than w_1^v. As a result, in the process of trying to correct the imbalance with respect to the second weight, we may end up worsening the imbalance with respect to the first weight. In fact, in [6] it has been shown that a scheme is not guaranteed to reach to a feasible solution.

For this reason, we have developed a different extension of the FM algorithm called FM2, that is better suited for refining a bisection in the presence of multiple vertex weights. This algorithm was originally proposed for the multi-constraint graph partitioning problem [6]. In FM2, instead of maintaining one priority queue we maintain m

queues for each one of the two partitions, where m is the number of weights. A vertex belongs to only a single priority queue depending on the relative order of the weights in its weight vector. In particular, a vertex v with weight vector $(w_1^v, w_2^v, \ldots, w_m^v)$, belongs to the jth queue if $w_j^v = \max_i(w_i^v)$. Given these $2m$ queues, the algorithm starts by initially inserting all the vertices to the appropriate queues according to their gains. Then, the algorithm proceeds by selecting one of these $2m$ queues, picking the highest gain vertex from this queue, and moving it to the other partition. The queue is selected as follows. If the current bisection represents a feasible solution, then the queue that contains the highest gain vertex among the $2m$ vertices at the top of the priority queues is selected. On the other hand, if the current bisection is infeasible, then the queue is selected depending on the relative weights of the two partitions. Specifically, if A and B are the two partitions, then the algorithm selects the queue corresponding to the largest w_i^x with $x \in \{A, B\}$ and $i = 1, 2, \ldots, m$. If it happens that the selected queue is empty, then the algorithm selects a vertex from the non-empty queue corresponding to the next heaviest weight *of the same partition*. For example, if $m = 3$ and

$$(w_1^A, w_2^A, w_3^A) = (.43, .60, .52) \quad \text{and} \quad (w_1^B, w_2^B, w_3^B) = (.57, .4, .48),$$

the algorithm will select the second queue of partition A. If this queue is empty, it will then try the third queue of A, followed by the first queue of A. Note that we give preference to the third queue of A as opposed to the first queue of B, even though B has more of the first weight than A does of the third. This is because our goal is to reduce the second weight of A. If the second queue of A is non-empty, we will select the highest gain vertex from that queue and move it to B. However, if this queue is empty, we still will like to decrease the second weight of A, and the only way to do that is to move a node from A to B. This is why when our first-choice queue is empty, we then select the most promising node from the same partition that this first-queue belongs to.

4 Experimental Results

We experimentally evaluated the quality of the partitionings produced by our multilevel multi-constraint hypergraph bisection algorithm on a set of two- and three-weight problems synthetically derived from the 18 hypergraphs that are part of the ISPD98 circuit partitioning benchmark suite [1]. The characteristics of these hypergraphs are shown in Table 1.

Benchmark	No. of vertices	No. of hyperedges
ibm01	12506	14111
ibm02	19342	19584
ibm03	22853	27401
ibm04	27220	31970
ibm05	28146	28446
ibm06	32332	34826
ibm07	45639	48117
ibm08	51023	50513
ibm09	53110	60902
ibm10	68685	75196
ibm11	70152	81454
ibm12	70439	77240
ibm13	83709	99666
ibm14	147088	152772
ibm15	161187	186608
ibm16	182980	190048
ibm17	184752	189581
ibm18	210341	201920

Table 1: The characteristics of the various hypergraphs used to evaluate the multilevel hypergraph partitioning algorithms.

Our starting point for deriving the synthetic multi-weight hypergraphs was the single-weight ISPD98 circuits that contain the actual areas for each one of the cell. For both our two- and three-weight problems, the first weight is always equal to the actual area of the cell. The second weight for each cell was set to be equal to the number of nets that each cell belongs to (*i.e.*, the incident degree of each cell). In the case of the three-weight problem, the third weight was set to be equal to the fan-out of each cell. We obtained that by assuming that the starting cell for each net in the ISPD98

circuits, is the driving cell for this net.

In all of our experiments, we set balance tolerances to be 45-55 for each of the different vertex weights, that is $[l_i, u_i] = [.45, .55]$ for $i = 1, 2, 3$. We performed all of our experiments on a 450MHz Pentium III–based Linux workstation.

4.1 Comparison of Refinement Schemes

In our first set of experiments, we compare the performance of the two different multi-constraint refinement algorithms FM1 and FM2 described in Section 3.3. In order to isolate the effects of the multilevel paradigm, we performed these comparisons by using these algorithms to compute a bisection of the original hypergraph, without performing any coarsening.

Table 2 shows a variety of statistics for the two refinement algorithms for all the circuits of the ISPD98 benchmark for two- and three-constraint problems. For each circuit we performed 50 different runs using both the FM1 and FM2 refinement algorithms. The columns labeled "Min-Cut" show the minimum cut achieved whereas the columns labeled "Avg-Cut" show the average cut achieved over all these 50 different runs. To compare the relative performance of FM2 over FM1, we computed the statistics shown in the last two columns of Table 2. These columns where computed by dividing the min-cut (average-cut) achieved by FM2 with the min-cut (average-cut) achieved by FM1. Any numbers lower than 1.0, indicate that FM2 performs better than FM1. Finally, the row labeled "ARQ" shows the Average Relative Quality of FM2 relative to FM1, and was obtained by averaging the values on the respective columns.

As we can see from this table, FM2 produces results that are better than those produced by FM1. In particular, on the average, FM2 performs 2% better with respect to the minimum cut, and 9% better with respect to the average cut. Looking at the individual problem instances we can see that with respect to the minimum cut, FM1 does at least 10% worse than FM2 in 9 out of the 36 instances, whereas FM2 does at least 10% worse in 7 instances. Similarly, with respect to the average cut, FM1 does at least 10% worse than FM2 in 19 instances, whereas FM2 does at least 10% worse in 0 instances.

4.2 Comparison of Coarsening Schemes

In our second set of experiments, we compare the performance of the two different coarsening schemes FC and BFC described in Section 3.1. In these experiments we used FM2 as the multi-constraint refinement algorithm in the multilevel framework. Table 3 shows a variety of statistics for the two coarsening schemes. For each circuit we performed 10 different runs using both the FC and BFC refinement algorithms.

As we can see from this table, there is little difference between the two coarsening schemes. BFC tends to produce bisections that are somewhat better than those produced by FC, but the difference is quite small. In particular, on the average BFC performs 2% better with respect to the minimum cut, and 0% better with respect to the average cut. Looking at the individual problem instances we can see that with respect to the minimum cut, FC does at least 10% worse than BFC in 3 out of the 36 instances, whereas BFC does at least 10% worse in 1 instances. Similarly, with respect to the average cut, FC does at least 10% worse than BFC in 3 instances, whereas BFC does at least 10% worse in 4 instances.

4.3 Comparison of Multilevel vs Single-level Partitioners

In our third set of experiments, we compare the performance of the FM2-based single-level multi-constraint partitioning algorithm against the multilevel multi-constraint partitioning algorithm that uses BFC for coarsening and FM2 for refinement. Table 4 shows a variety of statistics for the two partitioning algorithms. For each circuit we performed 50 different runs of the single-level partitioning algorithm and 10 different runs of the multilevel partitioning algorithm. Note that the columns labeled "Time" shows the total amount of time in seconds, required to compute all the different bisections.

As we can see from this table, the multilevel multi-constraint partitioning algorithm performs substantially better than the single-level partitioning algorithm. In particular, on the average the multilevel algorithm performs 29%

Circuit	NCon	FM1		FM2		FM2 relative to FM1	
		Min-Cut	Avg-Cut	Min-Cut	Avg-Cut	Min-Cut	Avg-Cut
ibm01	2	353	585.00	318	534.60	0.90	0.91
ibm01	3	515	795.20	421	659.20	0.82	0.83
ibm02	2	342	730.90	344	565.60	1.01	0.77
ibm02	3	423	827.70	314	634.30	0.74	0.77
ibm03	2	1051	1799.00	1072	1761.60	1.02	0.98
ibm03	3	1176	2165.50	1091	1837.50	0.93	0.85
ibm04	2	962	1714.50	871	1409.40	0.91	0.82
ibm04	3	1008	1842.70	945	1608.20	0.94	0.87
ibm05	2	2185	3572.70	2763	3541.20	1.26	0.99
ibm05	3	2099	3367.90	2333	3356.70	1.11	1.00
ibm06	2	972	1297.20	999	1370.30	1.03	1.06
ibm06	3	1025	1413.60	1045	1515.10	1.02	1.07
ibm07	2	1319	2212.30	1237	2139.40	0.94	0.97
ibm07	3	1241	2432.70	1317	2545.70	1.06	1.05
ibm08	2	1815	4394.20	1734	3214.10	0.96	0.73
ibm08	3	1832	3854.30	1890	2849.50	1.03	0.74
ibm09	2	1039	2711.70	1170	2248.10	1.13	0.83
ibm09	3	1627	3609.00	1663	3122.30	1.02	0.87
ibm10	2	1486	2534.90	1767	2581.00	1.19	1.02
ibm10	3	1975	3339.70	1691	2920.90	0.86	0.87
ibm11	2	1426	3657.80	1157	3196.10	0.81	0.87
ibm11	3	2089	4971.70	2039	4545.20	0.98	0.91
ibm12	2	2566	3656.60	2535	3664.70	0.99	1.00
ibm12	3	2725	5085.50	2713	3954.60	1.00	0.78
ibm13	2	1331	2261.80	1352	2422.10	1.02	1.07
ibm13	3	1330	3552.20	1451	3452.30	1.09	0.97
ibm14	2	5244	10534.90	3833	9471.40	0.73	0.90
ibm14	3	3224	7080.90	3592	6457.90	1.11	0.91
ibm15	2	4955	7994.90	4583	7690.50	0.92	0.96
ibm15	3	6511	9631.00	4842	9470.00	0.74	0.98
ibm16	2	2676	6779.40	3171	6091.50	1.18	0.90
ibm16	3	5441	9103.60	4309	7811.50	0.79	0.86
ibm17	2	3596	9066.10	3604	8467.80	1.00	0.93
ibm17	3	3921	10818.50	4873	9013.60	1.24	0.83
ibm18	2	3362	8221.50	3475	7701.50	1.03	0.94
ibm18	3	5420	8252.60	4056	8394.30	0.75	1.02
ARQ						0.98	0.91

Table 2: The performance of the FM1 and FM2 multi-constraint refinement algorithms for computing a bisection for hypergraphs with two and three constraints. For each scheme we show the minimum and average hyperedge cut achieved in 50 different runs. The columns labeled "FM2 relative to FM1" computes the relative minimum and average cut achieved by FM2 relative to that achieved by FM1. The row labeled 'ARQ' shows the Average Relative Quality of FM2 relative to FM1. For example, the ARQ value of 0.91 for the average cut, indicates that FM2 produces partitionings that on the average they cut 9% fewer hyperedges than those produced by FM1. The column labeled "NCon" indicates the number of balancing constraints.

better with respect to the minimum cut, and 52% better with respect to the average cut. Looking at the individual problem instances we can see that with respect to the minimum cut, the single-level algorithm does at least 10% worse than the multilevel algorithm in 31 out of the 36 instances, whereas the multilevel algorithm is never worse. Similarly, with respect to the average cut, the single-level algorithm does at least 10% worse than multilevel in all 36 instances. Note that these results are consistent with similar experiments comparing the multilevel and single-level partitioning algorithms for single constraint problems [1]. Finally, comparing the computational requirements of the two algorithms, we can see that the multilevel algorithm is significantly faster than the single-level FM2 algorithm. In fact comparing the total time required to partition all 36 problem instances, the single level algorithm is about 10 times slower.

4.4 Comparison of Multilevel Single- and Multi-Constraint Partitioners

In our last set of experiments, we compare the performance of our multilevel multi-constraint partitioning algorithm against the performance of hMETIS [5], which is a fast and high-quality single-constraint multilevel partitioning algorithm. Of course, hMETIS cannot compute multi-constraint partitionings, but when applied to the original single-constraint problem, it can provide us with some indications about the penalty associated when trying to balance multiple constraints.

Circuit	NCon	FC+FM2		BFC+FM2		BFC relative to FC	
		Min-Cut	Avg-Cut	Min-Cut	Avg-Cut	Min-Cut	Avg-Cut
ibm01	2	305	329.7	304	332.9	1.00	1.01
ibm01	3	311	343.9	297	325.4	0.95	0.95
ibm02	2	298	330.8	299	332.3	1.00	1.00
ibm02	3	310	362.7	297	355	0.96	0.98
ibm03	2	956	961.5	957	984	1.00	1.02
ibm03	3	956	963	963	1019.9	1.01	1.06
ibm04	2	680	725.9	686	818.6	1.01	1.13
ibm04	3	690	765	716	843	1.04	1.10
ibm05	2	1710	1762.2	1747	1776.6	1.02	1.01
ibm05	3	1710	1758.6	1751	1785.3	1.02	1.02
ibm06	2	937	980.6	967	1008	1.03	1.03
ibm06	3	943	1016.8	929	1009.6	0.99	0.99
ibm07	2	1032	1044.8	1000	1044.5	0.97	1.00
ibm07	3	1026	1050.4	1013	1049.3	0.99	1.00
ibm08	2	1332	1439.6	1324	1495.5	0.99	1.04
ibm08	3	1337	1475.5	1336	1431.1	1.00	0.97
ibm09	2	687	757.7	679	813.5	0.99	1.07
ibm09	3	684	736.6	684	795.7	1.00	1.08
ibm10	2	1496	1635.5	1418	1563.6	0.95	0.96
ibm10	3	1423	1589.3	1438	2013.1	1.01	1.27
ibm11	2	975	1189.7	974	1110.2	1.00	0.93
ibm11	3	966	1067.7	963	1039.4	1.00	0.97
ibm12	2	3180	3702.9	2522	3698.1	0.79	1.00
ibm12	3	2807	3550.5	2517	3115.5	0.90	0.88
ibm13	2	882	1074	855	1009.2	0.97	0.94
ibm13	3	876	1044.3	842	877.6	0.96	0.84
ibm14	2	2007	2156.9	1970	2214.5	0.98	1.03
ibm14	3	1945	2190	1961	2168.3	1.01	0.99
ibm15	2	3948	5645.1	2712	4372.2	0.69	0.77
ibm15	3	2712	4232.2	2771	3748.7	1.02	0.89
ibm16	2	1829	2723.7	2212	2902.2	1.21	1.07
ibm16	3	2113	2356.9	2100	2611.7	0.99	1.11
ibm17	2	2436	2499.2	2305	2538.1	0.95	1.02
ibm17	3	2390	2594	2337	2613	0.98	1.01
ibm18	2	1865	2036.8	1748	1932.9	0.94	0.95
ibm18	3	1875	2085.6	1818	2043.2	0.97	0.98
ARQ						0.98	1.00

Table 3: The performance of the multi-constraint bisection algorithm using the FC and BFC coarsening schemes.

Table 5 shows a variety of statistics for the two partitioning algorithms for all the circuits of the ISPD98 benchmark for one-, two-, and three-constraint problems. For each circuit we performed 10 different runs of the multilevel partitioning algorithms. For hMETIS we used the latest available version (version 1.5.3), and we used FC for coarsening, FM for refinement, and we did not perform any V-cycle refinement. For our multi-constraint algorithm, we used BFC for coarsening and FM2 for refinement. To compare the relative performance of multi- over single-constraint algorithms, we computed the statistics shown in the columns labeled "Rel-Min" and "Rel-Avg", in a fashion similar to that we did in Section 4.1.

As we can see from this table, the bisections that satisfy multiple balancing constraints tend to cut a larger number of hyperedges compared to the bisections that need to satisfy a single balancing constraint. In particular, for both the two and three constraint problems, the multi-constraint bisections cut about 30% more hyperedges. This should not be surprising, as computing a bisection that satisfies multiple balancing constraints is substantially harder than computing a single-constraint bisection. This is because as we increase the number of constraints, the feasible solution space becomes smaller as well as fragmented. Thus, there may be fewer high-quality bisections, and also due to the fragmentation, it may be harder to find them. Also, the quality of the multi-constraint relative to the single-constraint solution depends on how unbalanced the single-constraint solution is. For example, if the single-constraint bisections produced by hMETIS for ibm04, ibm06, ibm13, and ibm16 are used to induce a bisection for the two-constraint problem, then the balance with respect to the second weight is [.40, .60], [.01, .99], [.50, .50], and [.43, .57], respectively. As we can see, ibm06 is the most unbalanced, which is related to the large cut obtained by the multi-constraint algorithm. On the other hand, ibm13 happens to be balanced with respect to the second constraint, so the cut obtained by the multi-constraint algorithm is similar to that obtained by hMETIS.

Finally, comparing the computational requirements of the two algorithms, we can see that as the number of con-

Circuit	NCon	FM2			Multilevel BFC+FM2			Multilevel relative to FM2	
		Min-Cut	Avg-Cut	Time	Min-Cut	Avg-Cut	Time	Min-Cut	Avg-Cut
ibm01	2	318	534.60	70.32	304	332.90	14.08	0.96	0.62
ibm01	3	421	659.20	78.40	297	325.40	15.46	0.71	0.49
ibm02	2	344	565.60	132.84	299	332.30	30.70	0.87	0.59
ibm02	3	314	634.30	150.40	297	355.00	30.99	0.95	0.56
ibm03	2	1072	1761.60	200.96	957	984.00	29.99	0.89	0.56
ibm03	3	1091	1837.50	236.74	963	1019.90	34.89	0.88	0.56
ibm04	2	871	1409.40	218.65	686	818.60	35.94	0.79	0.58
ibm04	3	945	1608.20	257.76	716	843.00	36.85	0.76	0.52
ibm05	2	2763	3541.20	299.29	1747	1776.60	50.59	0.63	0.50
ibm05	3	2333	3356.70	341.99	1751	1785.30	52.98	0.75	0.53
ibm06	2	999	1370.30	306.71	967	1008.00	52.70	0.97	0.74
ibm06	3	1045	1515.10	345.39	929	1009.60	57.76	0.89	0.67
ibm07	2	1237	2139.40	442.58	1000	1044.50	70.84	0.81	0.49
ibm07	3	1317	2545.70	483.86	1013	1049.30	70.53	0.77	0.41
ibm08	2	1734	3214.10	976.96	1324	1495.50	89.31	0.76	0.47
ibm08	3	1890	2849.50	962.48	1336	1431.10	105.98	0.71	0.50
ibm09	2	1170	2248.10	514.98	679	813.50	81.45	0.58	0.36
ibm09	3	1663	3122.30	581.44	684	795.70	85.70	0.41	0.25
ibm10	2	1767	2581.00	801.56	1418	1563.60	116.79	0.80	0.61
ibm10	3	1691	2920.90	898.22	1438	2013.10	131.27	0.85	0.69
ibm11	2	1157	3196.10	873.34	974	1110.20	107.09	0.84	0.35
ibm11	3	2039	4545.20	953.51	963	1039.40	107.42	0.47	0.23
ibm12	2	2535	3664.70	785.46	2522	3698.10	107.60	0.99	1.01
ibm12	3	2713	3954.60	832.42	2517	3115.50	122.82	0.93	0.79
ibm13	2	1352	2422.10	1006.28	855	1009.20	132.88	0.63	0.42
ibm13	3	1451	3452.30	1131.50	842	877.60	129.88	0.58	0.25
ibm14	2	3833	9471.40	2592.91	1970	2214.50	273.04	0.51	0.23
ibm14	3	3592	6457.90	2444.16	1961	2168.30	324.99	0.55	0.34
ibm15	2	4583	7690.50	2857.68	2712	4372.20	293.24	0.59	0.57
ibm15	3	4842	9470.00	3049.41	2771	3748.70	330.78	0.57	0.40
ibm16	2	3171	6091.50	2966.12	2212	2902.20	363.68	0.70	0.48
ibm16	3	4309	7811.50	3279.86	2100	2611.70	409.37	0.49	0.33
ibm17	2	3604	8467.80	3640.84	2305	2538.10	502.02	0.64	0.30
ibm17	3	4873	9013.60	3733.47	2337	2613.00	567.93	0.48	0.29
ibm18	2	3475	7701.50	6261.56	1748	1932.90	473.90	0.50	0.25
ibm18	3	4056	8394.30	6327.64	1818	2043.20	523.32	0.45	0.24
ARQ								0.71	0.48
Tot. Time				51037.69			5964.76		

Table 4: The performance of the single-level and multi-level multi-constraint bisection algorithms.

straints increases, the overall amount of time required to compute the bisections also increases. However, this increase is quite small. Comparing the total amount of time to bisect all 18 circuits, the two- and three-constraint partitioners require 17% and 30% more time than the single-constraint algorithm, respectively.

References

[1] C. J. Alpert. The ISPD98 circuit benchmark suite. In *Proc. of the Intl. Symposium of Physical Design*, pages 80–85, 1998.

[2] C. J. Alpert, J. H. Huang, and A. B. Kahng. Multilevel circuit partitioning. In *Proc. of the 34th ACM/IEEE Design Automation Conference*, 1997.

[3] Charles J. Alpert and Andrew B. Kahng. Recent directions in netlist partitioning. *Integration, the VLSI Journal*, 19(1-2):1–81, 1995.

[4] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *In Proc. 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[5] G. Karypis and V. Kumar. hMeTiS 1.5: A hypergraph partitioning package. Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL *http://www.cs.umn.edu/~metis*.

[6] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of Supercomputing*, 1998. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[7] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *Proceedings of the Design and Automation Conference*, 1999.

[8] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. *IEEE Transactions on VLSI Systems*, 20(1), 1999. A short version appears in the proceedings of DAC 1997.

[9] T. Lengauer. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Winston, Boston, MA, 1976.

	hMeTiS 1.5.3 1 Constraint			Multilevel BFC+FM2 2 Constraints					Multilevel BFC+FM2 3 Constraints				
Circuit	Min Cut	Avg Cut	Time	Min Cut	Avg Cut	Time	Rel Min	Rel Avg	Min Cut	Avg Cut	Time	Rel Min	Rel Avg
ibm01	240	302.70	11.88	304	332.90	14.08	1.27	1.10	297	325.40	15.46	1.24	1.07
ibm02	293	294.80	19.19	299	332.30	30.70	1.02	1.13	297	355.00	30.99	1.01	1.20
ibm03	799	829.40	24.44	957	984.00	29.99	1.20	1.19	963	1019.90	34.89	1.21	1.23
ibm04	454	530.30	29.66	686	818.60	35.94	1.51	1.54	716	843.00	36.85	1.58	1.59
ibm05	1725	1761.50	39.97	1747	1776.60	50.59	1.01	1.01	1751	1785.30	52.98	1.02	1.01
ibm06	374	429.90	33.96	967	1008.00	52.70	2.59	2.34	929	1009.60	57.76	2.48	2.35
ibm07	863	880.70	53.16	1000	1044.50	70.84	1.16	1.19	1013	1049.30	70.53	1.17	1.19
ibm08	1158	1206.10	68.63	1324	1495.50	89.31	1.14	1.24	1336	1431.10	105.98	1.15	1.19
ibm09	541	646.00	56.16	679	813.50	81.45	1.26	1.26	684	795.70	85.70	1.26	1.23
ibm10	809	1044.80	91.65	1418	1563.60	116.79	1.75	1.50	1438	2013.10	131.27	1.78	1.93
ibm11	705	862.10	79.33	974	1110.20	107.09	1.38	1.29	963	1039.40	107.42	1.37	1.21
ibm12	2006	2270.50	124.96	2522	3698.10	107.60	1.26	1.63	2517	3115.50	122.82	1.25	1.37
ibm13	887	1017.00	99.13	855	1009.20	132.88	0.96	0.99	842	877.60	129.88	0.95	0.86
ibm14	1539	1748.30	236.30	1970	2214.50	273.04	1.28	1.27	1961	2168.30	324.99	1.27	1.24
ibm15	2193	2455.20	233.35	2712	4372.20	293.24	1.24	1.78	2771	3748.70	330.78	1.26	1.53
ibm16	1711	1758.90	331.82	2212	2902.20	363.68	1.29	1.65	2100	2611.70	409.37	1.23	1.48
ibm17	2286	2475.80	456.78	2305	2538.10	502.02	1.01	1.03	2337	2613.00	567.93	1.02	1.06
ibm18	1523	1893.30	431.19	1748	1932.90	473.90	1.15	1.02	1818	2043.20	523.32	1.19	1.08
ARQ							1.30	1.34				1.30	1.32
Tot. Time			2421.56			2825.84					3138.92		

Table 5: The performance of the single- and multi-constraint multilevel bisection algorithms.

[10] B. Mobasher, N. Jain, E.H. Han, and J. Srivastava. Web mining: Pattern discovery from world wide web transactions. Technical Report TR-96-050, Department of Computer Science, University of Minnesota, Minneapolis, 1996.

[11] S. Shekhar and D. R. Liu. Partitioning similarity graphs: A framework for declustering problmes. *Information Systems Journal*, 21(4), 1996.

[12] Sverre Wichlund and Einar J. Aas. On Multilevel Circuit Partitioning. In *Intl. Conference on Computer Aided Design*, 1998.