Chapter 1

MULTILEVEL HYPERGRAPH PARTITIONING

George Karypis
University of Minnesota, Department of Computer Science, Minneapolis, MN 55455
karypis@cs.umn.edu

Introduction

Hypergraph partitioning is an important problem with extensive application to many areas, including VLSI design [Alpert and Kahng, 1995], efficient storage of large databases on disks [Shekhar and Liu, 1996], and data mining [Mobasher et al., 1996, Karypis et al., 1999b]. The problem is to partition the vertices of a hypergraph into k equal-size parts, such that the number of hyperedges connecting vertices in different parts is minimized.

During the course of VLSI circuit design and synthesis, it is important to be able to divide the system specification into clusters so that the inter-cluster connections are minimized. This step has many applications including design packaging, HDL-based synthesis, design optimization, rapid prototyping, simulation, and testing. Many rapid prototyping systems use partitioning to map a complex circuit onto hundreds of interconnected FPGAs. Such partitioning instances are challenging because the timing, area, and I/O resource utilization must satisfy hard device-specific constraints. For example, if the number of signal nets leaving any one of the clusters is greater than the number of signal pins available in the FPGA, then this cluster cannot be implemented using a single FPGA. In this case, the circuit needs to be further partitioned, and thus implemented using multiple FPGAs.

Circuits can be naturally represented using hypergraphs. The hypergraph's vertices will represent the cells of the circuit, and the hyperedges will represent the nets connecting these cells. This mapping from a circuit to a hypergraph is illustrated in Figure 1.1. A high quality hypergraph partitioning algorithm greatly affects the feasibility, quality, and cost of the resulting system.

The importance of the problem has attracted a considerable amount of research interest and over the last thirty years a variety of heuristic algorithms

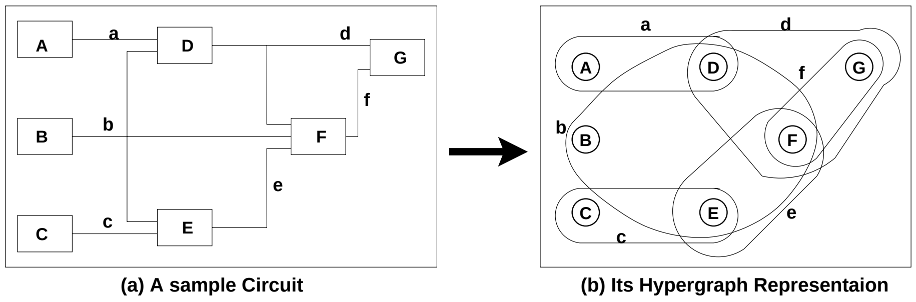


Figure 1.1. (a) A sample circuit drawn using cells and nets and (b) its hypergraph representation.

have been developed that offer different cost-quality trade-offs. The focus of this chapter is to describe one class of such algorithms that have come to be known as the ***multilevel partitioning algorithms***.

In recent years, multilevel partitioning algorithms have become the standard approach for partitioning large and irregular hypergraphs, as they provide high quality solutions, can scale to very large hypergraphs, and require a relatively small amount of time. These algorithms were initially developed for partitioning large graphs derived from scientific computations [Hendrickson and Leland, 1994, Karypis and Kumar, 1998b, Walshaw et al., 1997], but their advantages were quickly recognized by the VLSI CAD community, and a number of different multilevel algorithms have been developed [Karypis et al., 1999a, Alpert et al., 1997, Karypis and Kumar, 2000]. In this chapter we try to provide an overview of the multilevel paradigm and describe the various algorithms that it uses and why it works. Even though our presentation will be generic, from time-to-time we will use our experience in developing the hMETIS [Karypis and Kumar, 1998a] hypergraph partitioning package to discuss some of the finer details of successful implementations of multilevel algorithms.

1. Hypergraph Partitioning — Problem Definition

Formally, a hypergraph $G = (V, E)$ is defined as a set of vertices V and a set of hyperedges E. Each hyperedge is a subset of the set of vertices V [Berge, 1976], and the *size* of a hyperedge is the cardinality of this subset. We will assume that each vertex and hyperedge has a weight associated with it, and we will use $w(v)$ to denote the weight of a vertex v, and $w(e)$ to denote the weight of a hyperedge e. In the case of circuits, vertex weights usually represent the area of the corresponding module, and hyperedge weights represent some measure of criticality. When the original hypergraph is unweighted, both vertex- and hyperedge-weights are assumed to be one.

The goal of the k-way hypergraph partitioning problem is to partition the vertices of the hypergraph into k disjoint subsets $V_1, V_2, \ldots, V_k$, such that a certain function defined over the hyperedges is optimized, under the requirement that the size of each one of the partitions (*i.e.*, vertex subsets) is bounded (both from below and above). The requirement that the size of each partition is bounded is referred to as the ***partitioning constraint***, and the requirement that a certain function is optimized is referred to as the ***partitioning objective***.

Partitioning Constraints. There are a number of ways of specifying the partitioning constraint. A common way of doing this, especially in the context of hypergraph bi-partitioning, is to specify a lower-bound l and an upper-bound u on the size of each partition, such that $l + u = 1.0$. The goal then is to compute a bi-partitioning such that the sum of the weight of the vertices assigned to each partition $w(V_i)$ is bounded by

$$lw(V) \leq w(V_i) \leq uw(V),$$

where $w(V)$ is the sum of the weights of all the vertices. This approach can be extended to the k-way partitioning problem, by either using the same set of $[l, u]$ size-bounds for each partition, or specifying a different set $[l_i, u_i]$ size-bounds for each partition. However, in order for the constraints to be feasible, the sum of the upper-bounds over all the partitions should be greater than 1.0.

An alternate way of specifying a balance constraint, that is well-suited for k-way partitioning, is to supply an overall load imbalance tolerance c such that $c \geq 1.0$. In this case, the goal is to compute a partitioning such that the sum of the weight of the vertices $w(V_i)$ assigned to each partition V_i is bounded by

$$\frac{w(V)}{ck} \leq w(V_i) \leq \frac{cw(V)}{k}.$$

Partitioning Objectives. Over the years, a number of partitioning objective functions have been developed. The survey by Alpert and Kahng [Alpert and Kahng, 1995] provides a comprehensive description of a variety of objective functions that are commonly used for hypergraph partitioning in the context of VLSI design.

One of the most commonly used objective function is to ***minimize the hyperedge-cut*** of the partitioning; *i.e.*, the sum of the weights of the hyperedges that span multiple partitions. Another objective that is often used is to ***minimize the sum of external degrees*** (SOED) of all hyperedges that span multiple partitions. Given a k-way partitioning and a hyperedge e, the external degree of e is defined to be 0, if e is not cut by the partitioning, otherwise it is equal to the number of partitions that is spanned by e times its hyperedge weight. Then, the goal of the partitioning algorithm is to compute a k-way partitioning that minimizes the sum of external degrees of the hyperedges. An objective related to SOED is to ***minimize the*** $(K-1)$ ***metric*** [Alpert and Kahng, 1995, Cong and Lim, 1998]. In the case of the $(K-1)$ metric, the cost of a hyperedge that spans K partitions is $(K-1)$, whereas for the SOED metric, the cost is K.

In our discussion throughout this chapter will focus on the problem of minimized the hyperedge cut, but the other objectives can be optimized in a similar fashion.

1.1 Extensions on the Basic Problem

In addition to the above single-constraint single-objective partitioning problem, a number of ***multi-objective*** and ***multi-constraint*** formulations have been proposed. The idea behind the multi-objective formulations is to compute a partitioning that simultaneously optimizes more than one objective. These objectives can be defined *locally*, on the hyperedges that are cut by the partitioning, or they can be defined *globally*, in terms of certain properties of the overall partitioning. Examples of such local objectives are the minimization of the hyperedge cut, the SOED metric, and/or the $(K-1)$ metric; whereas global objectives usually capture signal delays, placement and routing costs.

Multi-objective problems that combine both local and global objectives are particularly hard to optimize in an efficient and problem independent way. On the other hand, there are reasonably good approaches for optimizing multi-objective problems in which the different objectives are defined locally. In this chapter we will not discuss such problems because their formulation is independent of the multilevel paradigm, and the reader should refer to [Schloegel et al., 1999] for a discussion of the issues involved and a general framework of using single-objective partitioning algorithms to solve multi-objective problems that use multiple locally defined objectives.

Multi-constraint partitioning formulations were introduced to model problems in which the desired partitioning needs to balance multiple weights associated with the vertices of the hypergraph. In this model, a vector of weights is assigned to each vertex and the goal is to produce a partitioning 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). 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.

The multi-constraint graph bisection problem is formally defined as follows. 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 the partitioning 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. Finally, 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 the modeling ability.

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:

$$l_i \leq \sum_{\forall v \in V : P[v]=1} w_i(v) \leq u_i \quad \text{and} \quad l_i \leq \sum_{\forall v \in V : P[v]=2} w_i(v) \leq u_i, \tag{1.1}$$

where $i = 1, 2, \ldots, m$ representing the different vertex weights. The multi-constraint k-way partitioning problem is defined in a similar fashion.

1.2 Methods for Computing a k-way Partitioning

The most commonly used approach for computing a k-way partitioning is based on recursive bisectioning. In this approach, the overall k-way partitioning is obtained by initially bisecting the hypergraph to obtain a two-way partitioning. Then, each of these parts if further bisected to obtain a four-way partitioning, and so on. Assuming that k is a power of two, then the final k-way partitioning can be obtained in $\log(k)$ such steps (or after performing $k-1$ bisections. In the cases in which k is not a power of two, the above approach needs to be modified so that each bisectioning produces appropriate size partitions. For example, a five-way partitioning can be obtained by splitting the hypergraph into two parts, one containing roughly 2/5 of the hypergraph and the other 3/5. The smaller of these two partitions will be further bisected to obtain two of the final five partitions, whereas the largest will be further partitioned into three parts via recursive bisection.

An alternative way of computing the k-way partitioning is to do so directly. There are a number of advantages of computing the k-way partitioning directly, that were identified as early back as in the seminal work by Kernighan and Lin [Kernighan and Lin, 1970]. First, a recursive bisection algorithm does not allow for the direct optimization of objectives that depend on knowing how the hyperedges are partitioned across all k partitions. Some examples of such objectives are the SOED (described earlier), scaled cost, and absorption that are described in [Alpert and Kahng, 1995]. Second, a k-way partitioning algorithm is capable of enforcing tighter balancing constraints while retaining the ability to sufficiently explore the feasible solution space to optimize the partitioning objective. This is especially true when the partitioning solution must simultaneously satisfy multiple balancing constraints [Karypis and Kumar, 1998c]. Third, a method that obtains a k-way partitioning directly can potentially produce much better partitionings than a method that computes a k-way partitioning via recursive bisection. In fact, in the context of a certain classes of graphs it was shown that recursive bisectioning can be up to an $O(\log n)$ factor worst than the optimal solution [Simon and Teng, 1993].

2. The Multilevel Paradigm for Hypergraph Partitioning

The key idea behind the multilevel approach for hypergraph partitioning is fairly simple and straightforward. Multilevel partitioning algorithms, instead of trying to compute the partitioning directly in the original hypergraph, first obtain a sequence of successive approximations of the original hypergraph. Each one of these approximations represents a problem whose size is smaller than the size of the original hypergraph. This process continues until a level of approximation is reached in which the hypergraph contains only a few tens of vertices. At this point, these algorithms compute a partitioning of that hypergraph. Since the size of this hypergraph is quite small, even simple algorithms such as Kernighan-Lin (KL) [Kernighan and Lin, 1970] or Fiduccia-Mattheyses (FM) [Fiduccia and Mattheyses, 1982] lead to reasonably good solutions. The final step of these algorithms is to take the partitioning computed at the smallest hypergraph and use it to derive a partitioning of the original hypergraph. This is usually done by propagating the solution through the successive better approximations of the hypergraph and using simple approaches to further refine the solution.

In the multilevel partitioning terminology, the above process is described in terms of three phases. The ***coarsening phase***, in which the sequence of successively approximate hypergraphs (*coarser*) is obtained, the ***initial partitioning phase***, in which the smallest hypergraph is partitioned, and the ***uncoarsening and refinement phase***, in which the solution of the smallest hypergraph is *projected* to the next level finer graph, and at each level an iterative refinement algorithm such as KL or FM is used to further improve the quality of the partitioning. The various phases of multilevel approach in the context of hypergraph bisection are illustrated in Figure 1.2.

This paradigm was independently studied by Bui and Jones [Bui and Jones, 1993] in the context of computing fill-reducing matrix reordering, by Hen-

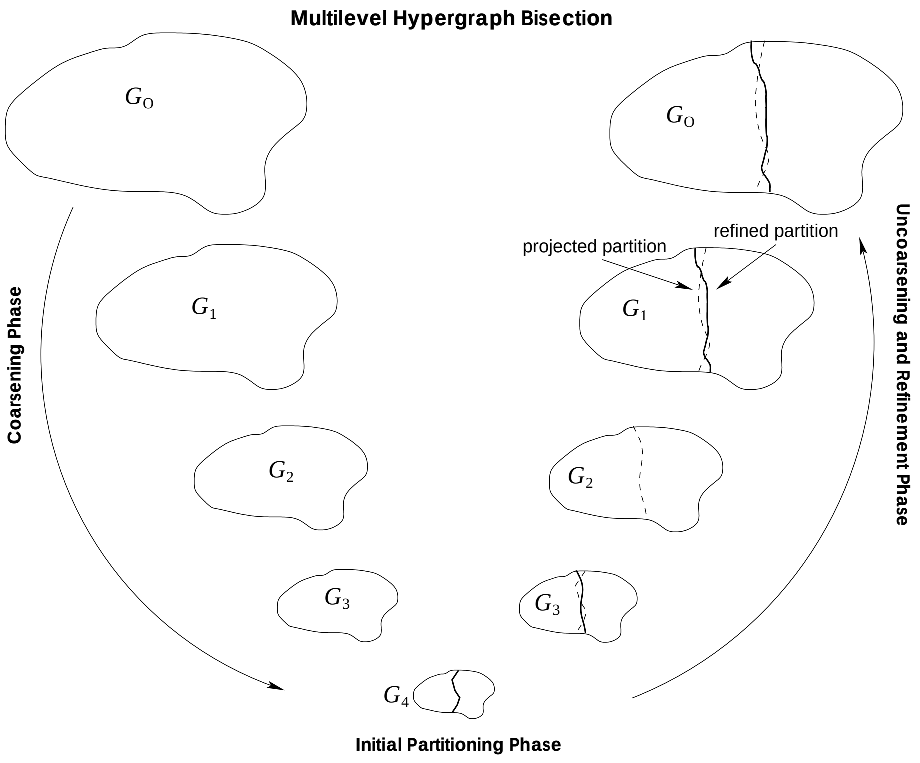

Multilevel Hypergraph Bisection
G_O
G_1
G_2
G_3
G_4
Coarsening Phase
projected partition
refined partition
Uncoarsening and Refinement Phase
Initial Partitioning Phase

Figure 1.2. The various phases of the multilevel hypergraph bisection. During the coarsening phase, the size of the hypergraph is successively decreased; during the initial partitioning phase, a bisection of the smaller hypergraph is computed; and during the uncoarsening and refinement phase, the bisection is successively refined as it is projected to the larger hypergraphs. During the uncoarsening and refinement phase, the dashed lines indicate projected partitionings and dark solid lines indicate partitionings that were produced after refinement.

drickson and Leland [Hendrickson and Leland, 1993] in the context of finite element mesh-partitioning, and by Hauck and Borriello [Hauck and Borriello, 1995] (called Optimized KLFM), and by Cong and Smith [Cong and Smith, 1993] for hypergraph partitioning. Karypis and Kumar extensively studied this paradigm in [Karypis and Kumar, 1998b, Karypis and Kumar, 1995, Karypis and Kumar, 1999a] for the partitioning of graphs. They presented novel graph coarsening schemes and they showed both experimentally and analytically that even a good bisection of the coarsest graph alone is already a very good bisection of the original graph. These coarsening schemes made the overall multilevel paradigm very robust and made it possible to use simplified variants of KL or FM refinement schemes during the uncoarsening phase, which significantly speeded up the refinement process without compromising overall quality. METIS [Karypis and Kumar, 1998b], a multilevel graph partitioning algorithm based upon this work, routinely finds substantially better partitionings than other popular techniques such as spectral-based partitioning algorithms [Pothen et al., 1990, Barnard and Simon, 1993], in a fraction of the time required by them. Karypis *et al* [Karypis et al., 1999a] extended their multilevel graph partitioning work to hypergraph partitioning. The hMETIS [Karypis and Kumar, 1998a] package contains many of these algorithms and have been shown to produce high-quality partitionings for a wide-range of circuits.

3. The Various Phases of the Multilevel Paradigm

We now turn our focus on describing in detail, the algorithms used in the various phases of the multilevel paradigm. Our discussion will primarily be based on how these phases are implemented in the popular hMETIS [1] hypergraph partitioning package, but whenever appropriate will also discuss various alternatives.

3.1 Coarsening Phase

During the coarsening phase, a sequence of successively smaller hypergraphs is constructed. The members of the sequence approximate the original hypergraph at successive coarser scales of resolution. This is probably the most important phase of the multilevel paradigm, and its overall success relies on being able to find reasonable methods for obtaining these coarser hypergraphs. We will refer to these methods as the *coarsening methods* or *schemes*.

There are two key requirements for a successful coarsening method:

1 Any partitioning in a coarse hypergraph can be easily translated (*i.e.*, ***projected***) to a partitioning of the next-level finer hypergraph.

2 The cut of the projected partitioning in the next-level finer hypergraph should be less or equal to the cut of the partitioning in a coarse hypergraph.

The first requirement is important as it allows us to easily propagate the partitioning to successive finer hypergraphs (all the way to the original problem), whereas the second requirement is essential to ensure that the refinement performed at each successive finer hypergraph is meaningful.

Recall from the general description of the multilevel paradigm (Section 2), that during the uncoarsening and refinement phase the solution of the coarsest hypergraph is used to induce a partitioning of the original hypergraph by propagating it to successive finer hypergraphs and refining it (*i.e.*, further optimiz-

[1] hMETIS is available on the WWW at http://www.cs.umn.edu/˜metis/hmetis.

ing) at each intermediate hypergraph. Now, this refinement at the intermediate hypergraphs will make sense if and only if any improvements in the quality of the partitioning at that level also translates to improvements of the corresponding partitioning of the original hypergraph. If that is not the case, then we may be spending a lot of effort just to make the solution worse.

Fortunately, there is a simple way to obtain successively coarse hypergraphs that automatically satisfy both of the above requirements. This is done by grouping the vertices of the hypergraph into disjoint clusters and collapsing the vertices of each cluster into a single vertex. Specifically, let $G = (V, E)$ be a hypergraph and let $C = \{c_1, c_2, \ldots, c_m\}$ be the set of the vertex-clusters such that for any pair of clusters $c_i \bigcap c_j = \emptyset$, $c_i \subset V$, and $\bigcup_i c_i = V$. Using this clustering, we then obtain a coarse hypergraph $G^c = (V^c, E^c)$ as follows. G^c will have exactly m vertices such that the ith vertex will correspond to the ith cluster of C. The weight of each vertex $v_i^c \in V^c$ will be set equal to the sum of the vertex weights of the vertices in c_i. Each hyperedge $e_j = \{v_{j,1}, v_{j,2}, \ldots, v_{j,k}\}$ of G will be mapped to a hyperedge e_j^c of G^c as follows. Let $M(v)$ be a function that maps each vertex $v \in V$ into its corresponding cluster in C and hence its corresponding vertex in G^c. Using this function, then e_j^c will be mapped to the hyperedge the contains the following vertices of G^c: $\{M(v_{j,1}), M(v_{j,2}), \ldots, M(v_{j,k})\}$. Now, since M is in general a many-to-one mapping, some of the vertices in the above set will occur multiple times and they are removed. Note that if after keeping only the unique vertices, the size of e_j^c becomes one, it is removed as it corresponds to a hyperedge that contains only one vertex. Also, it may be the case that different hyperedges in G get mapped to the same set of vertices in G^c. In that case, only one such hyperedge is kept, but its weight is set equal to the sum of the hyperedge weight of its corresponding set of hyperedges in G. The coarse graph obtained using the above method retains all essential information required to compute a partitioning that correctly models the balancing constraints and at the same time its cut will be identical to the cut of the corresponding partitioning of the original hypergraph.

Numerous approaches have been proposed for finding the groups of vertices to be merged together. Some of the underlying design goals of these schemes are the following.

1 The scheme should lead to coarser hypergraphs for which high-quality partitionings can readily be obtained. That is, the partition that can be obtained in the coarse hypergraph should not be significantly worse than the partition that can be obtained on the original hypergraph.

2 The coarsening should help in successively reducing the size of the hyperedges. That is, after several levels of coarsening, large hyperedges should have been contracted to hyperedges that connect just a few ver-

tices. This is particularly helpful, since refinement heuristics based on the KL and FM are very effective in refining small hyperedges but are quite ineffective in refining hyperedges with a large number of vertices belonging to different partitions.

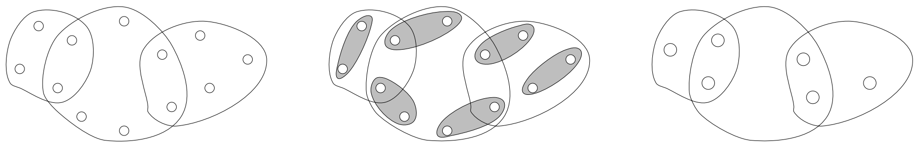
3 They should create successive hypergraphs in which the sum of the weight of the hyperedges reduces as quickly as possible. Recall from the above description, that each coarse hyperedge that gets mapped to a single coarse vertex is eliminated from the hypergraph. Consequently, a partitioning on that coarse hypergraph cannot possibly cut this hyperedge. Thus, one way for using the coarsening phase to help in improving our chances of finding a good partitioning, is to remove as many hyperedges as possible that can potentially be cut. We will refer to the sum of the weight of the hyperedges in the coarser hypergraphs as the *exposed hyperedge weight*, in contrast to the hyperedges that have been contracted into a single coarse vertex.

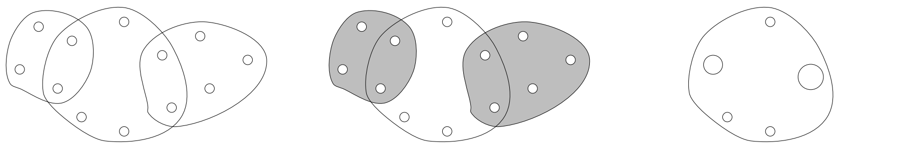
In the rest of this section we present a number of approaches for grouping vertices, followed by a discussion of some important aspects of the coarsening phase.

3.1.1 Edge Coarsening. The simplest way to group the vertices is to select pairs of vertices that are present in the same hyperedges, as illustrated in Figure 1.3(a). These pairs of vertices can be formed by finding a maximal matching of the vertices that are connected via hyperedges. A computationally efficient way of finding such a matching is to do it in randomized fashion as follows.

The vertices are visited in a random order. For each vertex v, all unmatched vertices that belong to hyperedges incident to v are considered, and one of them selected randomly is matched with v. Essentially this scheme computes a maximal matching on the graph representation of the hypergraph in which each hyperedge has been replaced by its clique representation [Lengauer, 1990]. However, this hypergraph-to-graph conversion is done implicitly during matching without forming the actual graph. For this reason, it is called the ***edge coarsening*** (EC) scheme.

The above basic randomized edge-selection scheme can be improved by realizing two facts associated with matchings and how they affect the coarse hypergraph they produce. First, there is in general a large number of matchings that have roughly the same cardinality. Second, since one of the goals of coarsening is to produce hypergraphs in which the exposed hyperedge weight is reduced as quickly as possible, we should give more emphasis on pairs of vertices that are part of a large number of smaller hyperedges, as these hyperedges can easily disappear in one or two coarsening steps. This is the basic idea behind the ***heavy-edge*** variation of the edge coarsening scheme, that instead of randomly selecting a vertex to match with v, it selects the unmatched vertex that is connected via the edge with the largest weight. The weight of an edge connecting two vertices v and u is computed as the sum of the *edge-weights* of all the hyperedges that contain v and u. Each hyperedge e of size $|e|$ is assigned an edge-weight of $1/(|e|-1)$, and as hyperedges collapse on each other during coarsening, their edge-weights are added up accordingly.


(a) Edge Coarsening


(b) Hyperedge Coarsening

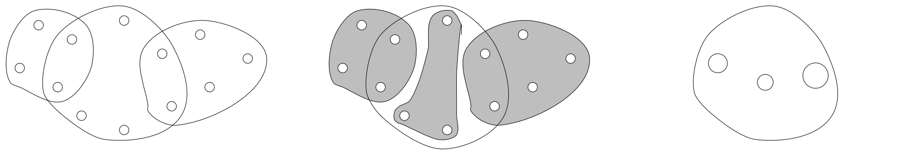

(c) Modified Hyperedge Coarsening

Figure 1.3. Various ways of matching the vertices in the hypergraph and the coarsening they induce. (a) In edge-coarsening, connected pairs of vertices are matched together. (b) In hyperedge-coarsening, all the vertices belonging to a hyperedge are matched together. (c) In modified hyperedge coarsening, we match together all the vertices in a hyperedge as well as all the groups of vertices belonging to a hyperedge.

3.1.2 Hyperedge Coarsening. Even though the edge coarsening scheme is able to produce successively coarser hypergraphs, it decreases the hyperedge weight of the coarser graph only for those pairs of matched vertices that are connected via a hyperedge of size two. As a result, the total hyperedge weight of successively coarser graphs does not decrease very fast. In order to ensure that for every group of vertices that are contracted together, there is a decrease in the hyperedge weight in the coarser graph, each such group of vertices must be connected by a hyperedge.

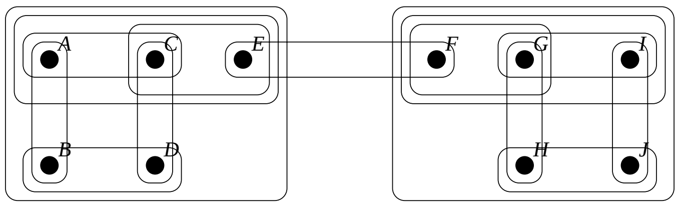
This is the motivation behind the ***hyperedge coarsening*** (HEC) scheme. In this scheme, an independent set of hyperedges is selected and the vertices that belong to individual hyperedges are contracted together, as illustrated in Figure 1.3(b). This is implemented as follows. The hyperedges are initially sorted in a non-increasing hyperedge-weight order and the hyperedges of the same weight are sorted in a non-decreasing hyperedge size order. Then, the hyperedges are visited in that order, and for each hyperedge that connects vertices that have not yet been matched, the vertices are matched together. Thus, this scheme gives preference to the hyperedges that have large weight and those that are of small size. After all of the hyperedges have been visited, the groups of vertices that have been matched are contracted together to form the next level coarser graph. The vertices that are not part of any contracted hyperedges are simply copied to the next level coarser graph.

3.1.3 Modified Hyperedge Coarsening. The hyperedge coarsening algorithm is able to significantly reduce the amount of hyperedge weight that is left exposed in successively coarser graphs. However, during each coarsening phase, a majority of the hyperedges do not get contracted because vertices that belong to them have been contracted via other hyperedges. This leads to two problems. First, the size of many hyperedges does not decrease sufficiently, making FM-based refinement difficult. Second, the weight of the vertices (*i.e.,* the number of vertices that have been collapsed together) in successively coarser graphs becomes significantly different, which distorts the shape of the contracted hypergraph.

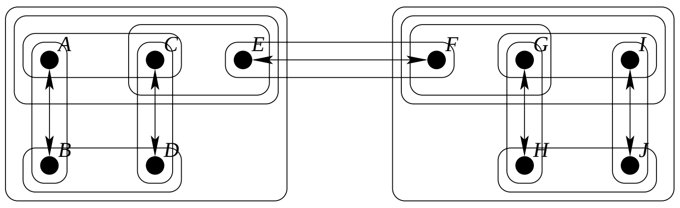
To correct this problem we implemented a ***modified hyperedge coarsening*** (MHEC) scheme as follows. After the hyperedges to be contracted have been selected using the hyperedge coarsening scheme, the list of hyperedges is traversed again. And for each hyperedge that has not yet been contracted, the vertices that do not belong to any other contracted hyperedge are contracted together, as illustrated in Figure 1.3(c).

3.1.4 FirstChoice Edge Coarsening. The edge- and the hyperedge-coarsening schemes share one characteristic that can potentially lead to less than ideal coarse representations of the original hypergraph. This common characteristic is that the grouping schemes employed by both approaches find maximal independent groups. That is, both the edge- and the hyperedge-coarsening schemes will find as many groups of vertices as they can, that are pair- or hyperedge-wise independent. The potential problem with this approach is that the independence (and to a certain degree, the maximality) requirement may destroy some clusters of vertices that naturally exist in the hypergraph.

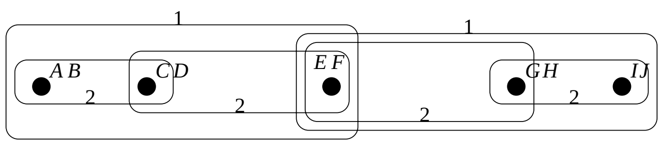
To see that, consider the example shown in Figure 1.4(a). As we can see from this figure, there are two natural clusters. The first cluster contains the five vertices on the left and the second cluster contains the five vertices on the right. These two clusters are connected by a single hyperedge; thus, the *natural* cut for this hypergraph is one. Figure 1.4(b) shows the pairs of vertices that are found by the edge-coarsening scheme. In the edge-coarsening scheme, vertex F will prefer to merge with vertex G, but vertex G had already been grouped with vertex H, consequently, vertex E is grouped with vertex F. Once the hypergraph is coarsened as shown in Figure 1.4(c), we can see that the natural separation point in this hypergraph has been eliminated, as it has been contracted in the vertex that resulted from merging E and F. A similar kind of example can be constructed using the hyperedge-coarsening as well.



(a) Initial Hypergraph



(b) Groups Determined by Edge−Coarsening



(c) Coarse Hypergraph

Figure 1.4. An example in which the edge-coarsening scheme can lead to a coarse representation in which the natural clusters of the hypergraph have been obscured. The weights on the hyperedges of the coarse hypergraph (c) represent the number of hyperedges in the original hypergraph that span the same set of vertices in the coarse representation.

The above observation led to develop of the ***FirstChoice*** (FC) coarsening scheme. The FC coarsening scheme is derived from the EC coarsening scheme by relaxing the requirement that a vertex is matched only with another unmatched vertex. Specifically, in the FC coarsening scheme, the vertices are again visited in a random order. However, for each vertex v, all vertices (both matched and unmatched) that belong to hyperedges incident to v are considered, and the one that is connected via the edge with the largest weight is matched with v, breaking ties in favor of unmatched vertices. As a result, each group of vertices to be merged together can contain an arbitrarily large number of vertices. The one potential problem with this coarsening scheme is that the number of vertices in successive coarse graphs may decrease by a large factor[2], potentially limiting the effect of refinement [Alpert et al., 1997]. Some methods of how to handle this type of problem are described in Section 3.1.6.

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 partitionings that require little refinement during the uncoarsening phase.

3.1.5 Coarsening for Multi-constraint Problems. The various coarsening schemes described so far can also be used to cluster the vertices in the context of multi-constraint hypergraph partitioning problems. This is because coarsening schemes that substantially reduce the exposed hyperedge weight and lead to coarse hypergraphs in which good partitionings can be computed, are equally critical for this problem, as well.

However, one can also use the coarsening process to try to reduce the inherently difficult load balancing problem that arises 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.

3.1.6 Additional Considerations for Coarsening Schemes. In addition to the exact method used to cluster together the vertices of the hypergraph, there is a number of other issues that need to be considered that, in general, impact the overall success of the multilevel paradigm.

[2]In the case of the EC coarsening scheme, the size of successive coarse graphs can be reduced by at most a factor of two.

The first issue has to do with the degree to which the various regions of hypergraph coarsen in a uniform way. Since, most of the above methods are randomized, there may be cases in which at successive coarser hypergraphs, the weight of certain vertices will increase at a higher rate than others. This usually happens because the coarsest vertices were obtained from larger clusters. In general this is not desirable because it significantly limits the number of partitionings of the coarsest graph that actually satisfy the balance constraints, consequently reducing our ability to optimize the cut. This problem is especially severe if the original hypergraph already has weights associated with the vertices, in which case there may be vertices at the coarsest level whose weight is greater than the upper bound on the weight of each partition. To prevent such pathological cases, all of the above coarsening schemes are usually modified so that they will never create clusters with large vertex weight. This can be done by either imposing an overall upper bound on the vertex-weight of the cluster, or imposing an upper bound that is progressively increasing with the coarsening level.

The second issue has to do with the rate at which we reduce the size of the hypergraphs during the coarsening phase. Schemes based on edge coarsening will in general reduce the size (in terms of vertices) of successive hypergraphs by a factor of two, whereas schemes based on (modified) hyperedge and first-choice can reduce the size by a much higher factor. One of the benefits of the multilevel paradigm is the fact that we perform refinement at different resolutions; thus, reducing the number of coarsening levels by quickly reducing the size of the successive coarse graphs may negatively affect the quality of the results. To address this problem, the various coarsening schemes can be easily modified to reduce the coarsening rate (by either selecting smaller groups or by finding non-maximal matches) and thus increasing the number of coarsening levels. However, even though this approach will tend to increase the quality of the partitionings, it will happen at the expense of higher memory and computational requirements. Experiments performed while developing hMETIS have shown that a reasonable balance between quality and runtime occurs when the size of successive graphs reduces by a factor in the range of 1.5–1.8.

Finally, one last issue that needs to be considered is when to actually stop the coarsening process. There are two competing factors that need to be considered. If we stop coarsening too early, then the coarsest hypergraph will be fairly large, as a result we may not be able to find a very good initial partitioning. On the other hand, if we stop the coarsening when there are only a very small number of vertices, the space of feasible solutions may be quite small, again reducing our ability to find a good solution. Thus, the right balance between these two factors needs to be established. Again, drawing from our experience with hMETIS, we found that a good point to stop the coarsening is when there are about $30k$ vertices, where k is the number of desired partitions.

3.2 Initial Partitioning Phase

During the initial partitioning phase, a partitioning of the coarsest hypergraph is computed, such that it minimizes the cut and satisfies the balancing constraints. Since this hypergraph has a very small number of vertices, the time to find a partitioning using any of the heuristic algorithms tends to be small. Note that it is not useful to find an optimal partition of this coarsest graph, as the initial partition will be substantially modified during the refinement phase.

Two algorithms have traditionally been used to compute the initial partitioning. The first algorithm simply creates a random partitioning that is *feasible*, in the sense that it satisfies the balancing constraints. The second algorithm starts from a randomly selected vertex and uses a breadth-first region-growing algorithm to *carve out* one of the partitions, and repeats this process for the rest of the partitions [Karypis and Kumar, 1999a]. After a partitioning is constructed using either of these algorithms, the partitioning is refined using the various refinement algorithms that are described in Section 3.3.

Since both algorithms are randomized, different runs give solutions of different quality. For this reason, a small number of different initial partitionings is commonly computed, and the one with the smallest cut is selected as the initial partitioning. A potential problem with this approach is that the partitioning of the coarsest hypergraph that has the smallest cut may not necessarily be the one that will lead to the smallest cut in the original hypergraph. It is possible that another partitioning of the coarsest hypergraph (with a higher cut) will lead to a better partitioning of the original hypergraph after the refinement is performed during the uncoarsening phase. For this reason, instead of selecting a single initial partitioning (*i.e.*, the one with the smallest cut), one can choose to propagate all or a subset of the initial partitionings.

Note that propagation of i initial partitionings increases the time during the refinement phase by a factor of i. Thus, by increasing the value of i, we can potentially improve the quality of the final partitioning at the expense of higher runtime. One way to dampen the increase in runtime due to large values of i is to eliminate unpromising partitionings as the hypergraph is uncoarsened. For example, one possibility is to propagate only those partitionings whose cuts are within x% of the best partitionings at the current level. If the value of x is sufficiently large, then all partitionings will be maintained and propagated in the entire refinement phase. On the other hand, if the value of x is sufficiently small, then on average only one partitioning will be maintained, as all other partitionings will be eliminated at the coarsest level. For moderate values of x, many partitionings may be available at the coarsest graph, but the number of such available partitionings will decrease as the graph is uncoarsened. This is useful for two reasons. First, it is more important to have many alternate partitionings at the coarser levels, as the size of the cut of a partitioning at a coarse level is a less accurate reflection of the size of the cut of the original finest level hypergraph. Second, refinement is more expensive at the fine levels, as these levels contain far more nodes than the coarse levels. Hence by choosing an appropriate value of x, we can benefit from the availability of many alternate partitionings at the coarser levels and avoid paying the high cost of refinement at the finer levels by keeping fewer candidates on average.

3.3 Uncoarsening and Refinement 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 reduce the cut-set (and thus to improve the quality of the partitioning) without violating the user specified balance constraints. Since the next level finer hypergraph has more degrees of freedom, such refinement algorithms tend to improve the solution quality.

The partitioning refinement approaches that have historically being used in the context of the multilevel paradigm are variations of the traditional algorithm by Fiduccia and Mattheyses, despite the fact that significantly more powerful algorithms such as the look-ahead scheme by Krishnamurthy [Krishnamurthy, 1984], and the PROP [Dutt and Deng, 1996a], and CLIP [Dutt and Deng, 1996b] schemes by Dutt and Deng. The primary reason for that is that the multilevel paradigm itself, by performing refinement at different levels, automatically offers some of the characteristics that the above schemes exploit.

3.3.1 Bisection Refinement Algorithms. The partitioning refinement algorithm by Fiduccia and Mattheyses is iterative in nature. It starts with an initial partitioning of the hypergraph. In each iteration, it tries to find subsets of vertices in each partition, such that moving them to other partitions improves the quality of the partitioning (*i.e.*, the number of hyperedges being cut decreases) and this does not violate the balance constraint. If such subsets exist, then the movement is performed and this becomes the partitioning for the next iteration. The algorithm continues by repeating the entire process. If it cannot find such a subset, then the algorithm terminates, since the partitioning is at a local minimum and no further improvement can be made by this algorithm.

For each vertex v, the FM algorithm computes the ***gain***, which is the reduction in the hyperedge-cut achieved by moving v to the other partition. 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 satisfies the balancing constraints or not. Note that the algorithm does not allow locked vertices to be moved, since this may result in thrashing (*i.e.*, repeated movement of the same vertex). A single pass of the algorithm ends when there are no more unlocked vertices. Then, the recorded values of the cut are checked, and the point where the minimum value was achieved while satisfying the balancing constraints, 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. With the use of appropriate data-structures, the complexity of each pass of the FM algorithm is $O(|E^h|)$ [Fiduccia and Mattheyses, 1982].

Multilevel Considerations. For refinement in the context of multilevel schemes, the initial partitioning obtained from the next level coarser graph is actually a very good partition. For this reason we can make a number of optimizations to the original FM algorithm. The first optimization limits the maximum number of passes performed by the FM algorithm to only two. This is because the greatest reduction in the cut is obtained during the first or second pass, and any subsequent passes only marginally improve the quality. Our experience has shown that this optimization significantly improves the runtime of FM without affecting the overall quality of the produced partitionings. The second optimization aborts each pass of the FM algorithm before actually moving all the vertices. The motivation behind this is that only a small fraction of the vertices being moved actually lead to a reduction in the cut, and after some point, the cut tends to increase as we move more vertices. When FM is applied to a random initial partitioning, it is quite likely that after a long sequence of *bad* moves, the algorithm will climb-out of a local minimum and reach to a better cut. However, in the context of a multilevel scheme, a long sequence of cut-increasing moves rarely leads to a better local minimum. For this reason, each pass of the FM algorithm can be stopped as soon as we have performed k vertex moves that did not improve the cut. Reasonable values of k are in the range of 1%–5% of the number of vertices in the hypergraph being refined. This modification to FM, called ***early-exit FM*** (FM-EE), does not significantly affect the quality of the final partitioning, but it dramatically improves the run time.

Another issue that also arises when refining the hypergraphs at the coarsest levels is that it may be very hard to perform any vertex moves and still maintain the balance constraints. This is especially true when the original hypergraphs have weights associated with the vertices and when the balancing constraints are tight. There are two things that are quite often beneficial in this type of situations. First, allow the solution to momentarily violate the balancing constraints as it moves vertices between the two partitions, but ensure that only solution instances that are actually balanced are recorded. Second, it may be beneficial to relax the balancing constraints at the coarsest levels and make them tighter incrementally as the solution gets propagated to successive finer hypergraphs.

3.3.2 k-way Refinement Algorithms. Refining a k-way partitioning is significantly more complicated because vertices can move from a partition to many other partitions; thus, increasing the optimization space combinatorially. An extension of the FM refinement algorithm in the case of k-way refinement is described in [Sanchis, 1989]. This algorithm uses $k(k-1)$ priority queues, one for each type of move. In each step of the algorithm, the moves with the highest gain are found from each of these $k(k-1)$ queues, and the move with the highest gain that preserves or improves the balance, is performed. After the move, all of the $k(k-1)$ priority queues are updated. The complexity of k-way refinement is significantly higher than that of 2-way refinement, and is only practical for small values of k. Furthermore, as the experiments in [Cong and Lim, 1998] suggest, the k-way FM algorithm is also very susceptible of being trapped into a local minimum that is far from being optimal.

The hill-climbing capability of the FM algorithm serves a very important purpose. It allows movement of an entire cluster of vertices across a partition boundary. Note that it is quite possible that as the cluster is moved across the partition boundary, the value of the objective function increases, but after the entire cluster of vertices moves across the partition, then the overall value of the objective function comes down. In the context of multilevel schemes, this hill-climbing capability becomes less important. The reason is that these clusters of vertices are coarsened into a single vertex at successive coarsening phases. Hence, movement of a vertex at a coarse level really corresponds to the movement of a group of vertices in the original hypergraph.

If the hill-climbing part of the FM algorithm is eliminated (*i.e.*, if vertices are moved only if they lead to positive gain), then it becomes less useful to maintain a priority queue. This is because vertices whose move results in a large positive gain will most likely be moved anyway even if they are not moved earlier (in the priority order). Hence, a variation of the FM algorithm that simply visits the vertices in a random order and moves them if they result in a positive gain is likely to work well in the multilevel context. Furthermore, the complexity of this algorithm will be independent of the number of partitions being refined, leading to a fast algorithm. For these reasons, hMETIS uses a ***greedy refinement*** algorithm to refine a k-way partitioning. It consists of a number of iterations. In each iteration all the vertices are checked to see if they can be moved so that the partitioning objective function is optimized, subject to the partitioning balancing constraint (as described in Section 2).

More precisely, the greedy k-way refinement algorithm works as follows. Consider a hypergraph $G = (V, E)$, and its partitioning vector P. The vertices are visited in a random order. Let v be such a vertex, let $P[v] = a$ be the partition that v belongs to. If v is a node internal to partition a then v is not moved. If v is at the boundary of the partition, then v can potentially be moved to one of the partitions $N(v)$ that vertices adjacent to v belong to (the set $N(v)$ is often refer to as the ***neighborhood*** of v). Let $N'(v)$ be the subset of $N(v)$ that contains all partitions b such that movement of vertex v to partition b does not violate the balancing constraint. Now the partition $b \in N'(v)$ that leads to the greatest positive reduction (gain) in the objective function is selected and v is moved to that partition.

The above greedy refinement algorithm can be used to compute a partitioning that minimizes a variety of objective functions, by appropriately computing the gain achieved in moving a vertex. Our current implementation allows a choice of two different objective functions. The first minimizes the hyperedge cut and the second minimizes the sum of external degrees (SOED) (Section 2).

Experiments with this greedy k-way refinement algorithm show that it converges after a small number of iterations and that it leads to reasonably good solutions [Karypis and Kumar, 1999b].

3.3.3 Multi-constraint Refinement Algorithms. The 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-constraint 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-constraint 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 such cases, a reasonable way for selecting the source partition is to choose 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 this approach will select B to be the source, as .60 is greater than .56 (that A contains with respect to the first weight). This scheme is referred to as FM1 [Karypis and Kumar, 1998c, Karypis, 1999].

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, it may end up worsening the imbalance with respect to the first weight. In fact, in [Karypis and Kumar, 1998c] it has been shown that a scheme is not guaranteed to reach to a feasible solution.

For this reason, a different extension of the FM algorithm called FM2 was proposed [Karypis and Kumar, 1998c, Karypis, 1999], that is better suited for refining a bisection in the presence of multiple vertex weights. In FM2, instead of maintaining one priority queue it maintains 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 $m = 3$,

$$(w_1(A), w_2(A), w_3(A)) = (.43, .60, .52),$$

and

$$(w_1(B), w_2(B), w_3(B)) = (.57, .4, .48),$$

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. Why Does the Multilevel Paradigm Work?

Extensive experimental studies have shown that multilevel graph partitioning algorithms are extremely robust and lead to high quality solutions, both for small and very large hypergraphs. In this section we attempt to explain the three primary reasons that explains the robustness of these algorithms.

Coarsening Makes the Problem Easier. A good coarsening scheme can hide a large number of the original hyperedges on the coarsest hypergraph. Figure 1.5 illustrates this point for simple graphs. The original graphs in Figures 1.5(a) and (b) have total edge weights of 37. After coarsening is performed on each, their total edge weights are reduced. Figures 1.5(a) and (b) show two possible coarsening heuristics, random and heavy-edge. In both cases, the total weight of the visible edges in the coarsened graph is less than that on the original graph. Note that by reducing the exposed edge weight, the task of computing a good quality partitioning becomes easier. For example, a worst case partitioning (*i.e.*, one that cuts every edge) of the coarsest graph will be of higher quality than the worst case partitioning of the original graph. Also, a random bisection of the coarsest graph will tend to be better than a random bisection of the original graph [Karypis and Kumar, 1995].

Multi-scale Refinement. Incremental refinement schemes such as KL/FM become much more powerful in the multilevel context. Here, the movement of a single vertex across the subdomain boundary in one of the coarse hypergraphs is equivalent to the movement of a large number of highly connected vertices in the original hypergraph, but is much faster. The ability of a refinement algorithm to move groups of highly connected vertices allows the algorithm to escape from some types of local minima. Figure 1.6 illustrates this phenomenon, again for simple graphs. The uncoarsened graph on the left hand side of Figure 1.6 is in a local minimum. However, the coarsened graph on the right side is not. Edge-cut reducing moves can be made here that will result in the left side graph escaping from its local minimum. Modifications of

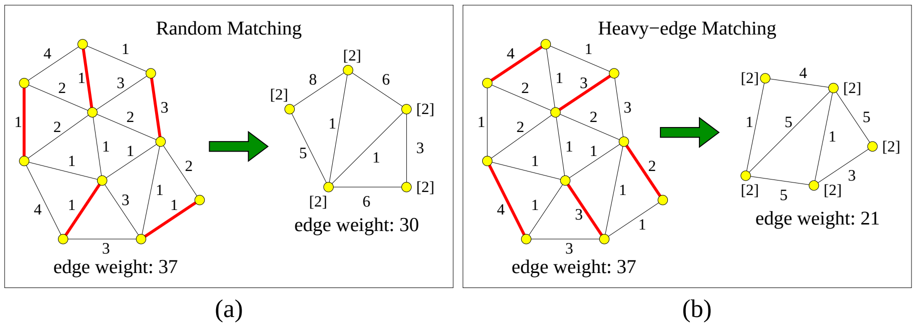


Figure 1.5. A random matching of a graph along with the coarsened graph (a). The same graph is matched (and coarsened) with the heavy-edge heuristic in (b). The heavy-edge matching minimizes the exposed edge weight.

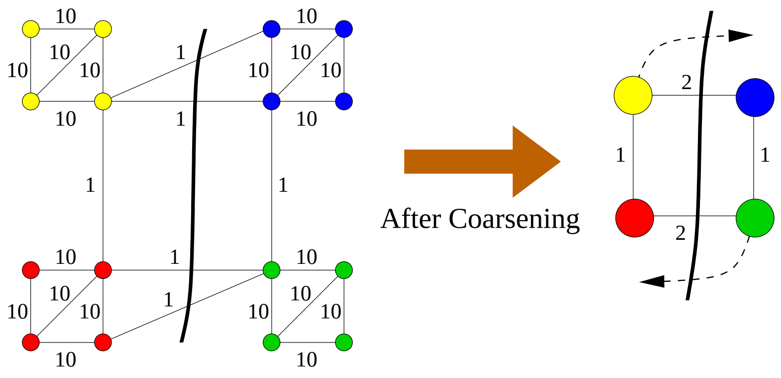


Figure 1.6. A partitioning in a local minimum for a graph and its coarsened variant. The graph to the right has been coarsened. Now it is possible to escape from the local minimum with edge-cut reducing moves.

KL/FM schemes have been developed that attempt to move sets of vertices in this way. However, computing these sets is computationally intensive. Multilevel schemes obtain much of the same benefits quickly.

Make the Refinement Problem Easier. The previous two reasons are equally applicable to graphs and hypergraphs. However, the third reason that can explain why the multilevel paradigm works so well is specify to hypergraphs. One of the challenges in refining the partitioning of a hypergraph using the various KL/FM algorithms is that large hyperedges tend to limit our ability to get out of local minima. This is illustrated in Figure 1.7(a) that shows a bisection that equally splits a large net of size six. In order for this net to be removed off the cut, the KL/FM algorithms need to move all three of the cells from partition one to partition two, without moving any of the cells

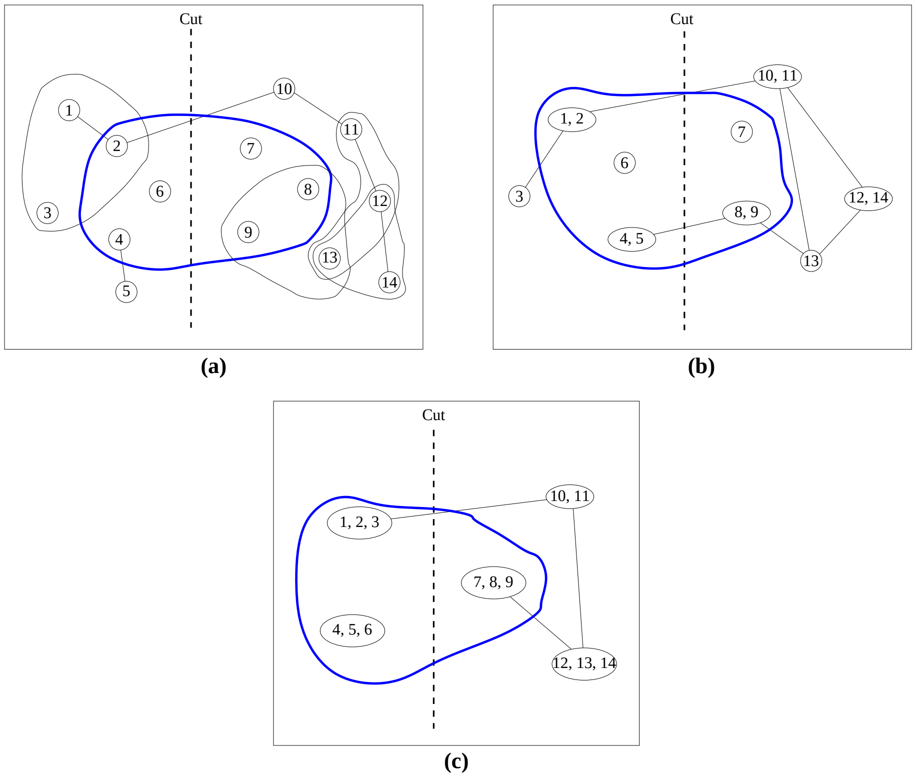


Figure 1.7. The effect of coarsening on the size of the hyperedges.

from two to one. However, this may be very hard to prevent using local gain information. Now in the context of the multilevel paradigm, as a result of coarsening, the size of these nets is progressively decreasing, as illustrated in Figures 1.7(b) and (c). Consequently, when the KL/FM refinement approaches are applied to coarser hypergraphs, they will be more effective as there will not be very many large nets.

5. Extensions of the Multilevel Paradigm

Although the multilevel paradigm is quite robust, randomization is inherent in all three phases of the algorithm. In particular, the random choice of vertices to be matched in the coarsening phase can disallow certain hyperedge-cuts, reducing the refinement possibilities in the uncoarsening phase. For example, consider the example hypergraph in Figure 1.8(a) and its two possible condensed versions (Figure 1.8(b) and 1.8(c)) with the same partitioning. The version in Figure 1.8(b) is obtained by selecting hyperedges *a* and *b* to be compressed in the hyperedge coarsening phase and then selecting pairs of nodes

(4,5), (6,7), and (8,9) to be compressed in the modified hyperedge coarsening phase. Similarly, the version shown in Figure 1.8(c) is obtained by selecting hyperedge *c* to be compressed in the hyperedge coarsening phase and then selecting pairs of nodes (6,7) and (8,9) to be compressed in the modified hyperedge coarsening phase. In the version of Figure 1.8(b) vertex *A*(4,5) can be moved from partition P_0 to P_1 to reduce the hyperedge-cuts by 1, but in Figure 1.8(c) no vertex can be moved to reduce the hyperedge-cuts.

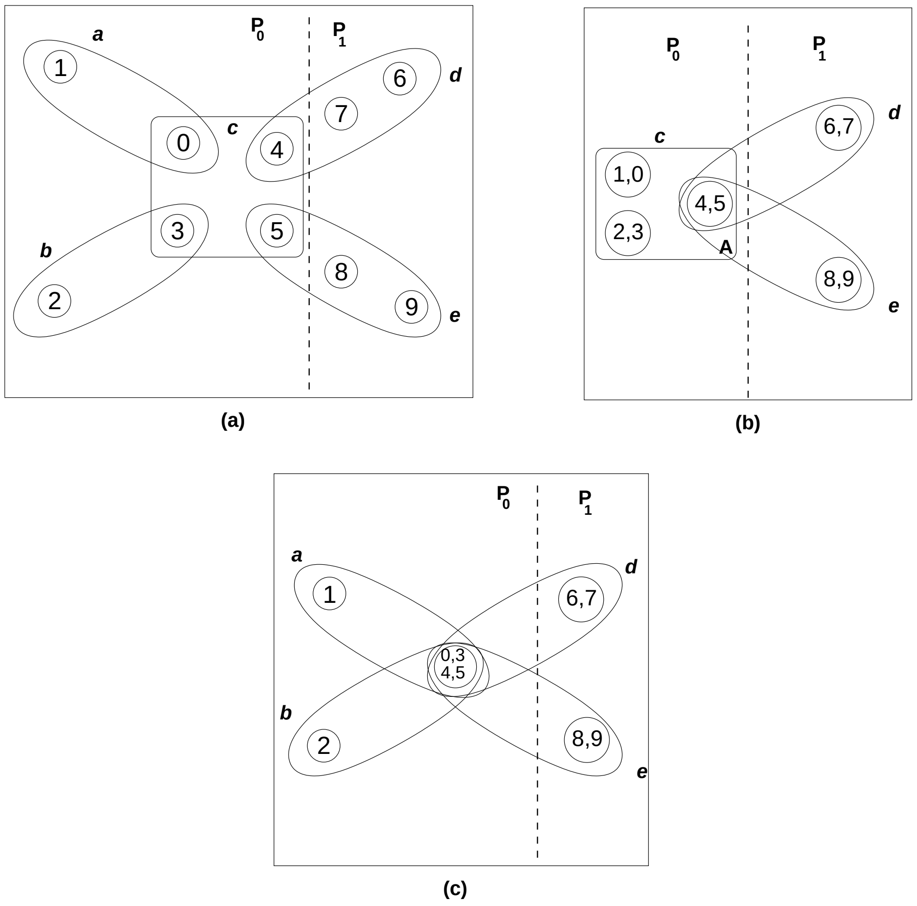


Figure 1.8. Effect of *Restricted coarsening*. (a) example hypergraph with a given partitioning with the required balance of 40/60, (b) a possible condensed version of (a), and (c) another condensed version of a hypergraph.

What this example shows is that in a multilevel setting, a given initial partitioning of a hypergraph can be potentially refined in many different ways depending upon how the coarsening is performed. Hence, a partitioning produced by a multilevel partitioning algorithm can be potentially further refined if the partitions are again coarsened in a manner different from the previous coarsening phase (which is easily done given the random nature of all of the coarsening schemes described here). The power of iterative refinement at different coarsening levels can also be used to develop a partitioning refinement algorithm based on the multilevel paradigm.

The idea behind this ***multi-phase refinement*** algorithm is quite simple. It consists of two phases, namely a coarsening and an uncoarsening phase. The uncoarsening phase of the multi-phase refinement algorithm is identical to the uncoarsening phase of the multilevel hypergraph partitioning algorithm described in Section 3.3. The coarsening phase however is somewhat different, as it preserves the initial partitioning that is input to the algorithm. We will refer to this as ***restricted coarsening*** scheme. Given a hypergraph G and a partitioning P, during the coarsening phase a sequence of successively coarser hypergraphs and their partitionings is constructed. Let (G_i, P_i) for $i = 1, 2, \dots, m$, be the sequence of hypergraphs and partitionings. Given a hypergraph G_i and its partitioning P_i, restricted coarsening will collapse vertices together that belong to only one of the two partitions. The partitioning P_{i+1} of the next level coarser hypergraph G_{i+1} is computed by simply inheriting the partition from G_i. For example, if a set of vertices $\{v_1, v_2, v_3\}$ from partition A is collapsed together to form vertex u_i of G_{i+1}, then vertex u_i belongs to partition A as well. By constructing G_{i+1} and P_{i+1} in this way we ensure that the number of hyperedges cut by the partitioning is identical to the number of hyperedges cut by P_i in G_i. The set of vertices to be collapsed together in this restricted coarsening scheme can be selected by using any of the coarsening schemes described in Section 3.1, namely edge-coarsening, hyperedge-coarsening, or modified hyperedge-coarsening.

Due to the randomization in the coarsening phase, successive runs of the multi-phase refinement algorithm can lead to additional reductions in the hyperedge cut. Thus, the multi-phase refinement algorithm can be performed iteratively. Note that during the refinement phase, we only propagate a single partitioning; thus, multi-phase refinement is quite fast.

In the context of the multilevel hypergraph partitioning algorithm, this new multi-phase refinement can be used in a number of ways.

V-Cycle. In this scheme, the best solution obtained from the multilevel partitioning algorithm (P_b) is improved using multi-phase refinement repeatedly. The multi-phase refinement stops when the solution quality cannot be improved further. The number of multi-phase refinement steps performed is problem dependent, and in general it increases as the size of the hypergraph increases. This is due to the larger solution space of the large hypergraphs.

v-Cycle. Our experience with the multilevel partitioning algorithm has shown that refining multiple solutions is expensive, especially during the final uncoarsening levels when the size of the contracted hypergraphs is large. One way to reduce the high cost of refining multiple solutions during the final uncoarsening levels is to select the best partitioning at some point in the uncoarsening phase and further refine only this best partitioning using multiphase refinement. This is the idea behind the v-cycle refinement.

In particular, let $G_{m/2}$ be the coarse hypergraph at the midpoint between G_0 (original hypergraph) and G_m (coarsest hypergraph). Let $P_{m/2}$ be the best partitioning at $G_{m/2}$. Then we use ($G_{m/2}$, $P_{m/2}$) as the input to multi-phase refinement. Since $G_{m/2}$ is relatively small as compared to G_m, multi-phase refinement converges in a small number of iterations. By using v-cycles, we can significantly reduce the amount of time spent in the refinement phase, especially for large hypergraphs. However, the overall quality can potentially decrease because we may have not picked up the best overall partitioning at $G_{m/2}$.

vV-Cycle. We can combine both V-cycles and v-cycles in the algorithm to obtain high quality partitioning in a small amount of time. In this scheme we use v-cycles to partition the hypergraph followed by the V-cycles to further improve the partition quality. V-cycles used in this way are particularly effective in significantly improving the hyperedge cut.

6. Direction of Future Research

Numerous studies have shown that multilevel graph partitioning algorithms are very successful in producing high quality hypergraph partitionings in a relatively small amount of time. It may be possible to further improve the quality of the partitionings produced by these algorithms in many ways. In particular, one area that is still not very well-understood, is the correct choice of the coarsening method. Our experience has been that no single coarsening method dominates the rest for all the different benchmarks. Further research is required to identify coarsening methods that are suitable for a wider class of hypergraphs.

The success of multilevel graph partitioning, has also sparked an increased interest in applying the key ideas of the multilevel paradigm to other hard combinatorial optimization problems arising in VLSI design. Many of the articles in this book describe some of these new algorithms, and the results being reported are encouraging, suggesting that the multilevel paradigm can potentially be applied to solve problems in placement, routing, and timing optimization. However, numerous research questions are still open regarding the properties of the correct coarsening and refinement methods for these new problems.

References

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

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

[Barnard and Simon, 1993] Barnard, S. T. and Simon, H. D. (1993). A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. In *Proceedings of the sixth SIAM conference on Parallel Processing for Scientific Computing*, pages 711–718.

[Berge, 1976] Berge, C. (1976). *Graphs and Hypergraphs*. American Elsevier, New york.

[Bui and Jones, 1993] Bui, T. and Jones, C. (1993). A heuristic for reducing fill in sparse matrix factorization. In *6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445–452.

[Cong and Lim, 1998] Cong, J. and Lim, S. K. (1998). Multiway Partitioning with Pairwise Movement. In *Intl. Conference on Computer Aided Design*.

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

[Dutt and Deng, 1996a] Dutt, S. and Deng, W. (1996a). A probability-based approach to VLSI circuit partitioning. In *Proceedings of the ACM/IEEE Design Automation Conference*.

[Dutt and Deng, 1996b] Dutt, S. and Deng, W. (1996b). VLSI circuit partitioning by cluster-removal using iterative improvement techniques. In *Proc. Physical Design Workshop*.

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

[Hauck and Borriello, 1995] Hauck, S. and Borriello, G. (1995). An evaluation of bipartitioning technique. In *Proc. Chapel Hill Conference on Advanced Research in VLSI*.

[Hendrickson and Leland, 1993] Hendrickson, B. and Leland, R. (1993). A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories.

[Hendrickson and Leland, 1994] Hendrickson, B. and Leland, R. (1994). The chaco user's guide, version 2.0. Technical Report SAND94-2692, Sandia National Laboratories.

[Karypis, 1999] Karypis, G. (1999). Multilevel algorithms for multi-constraint hypergraph partitioning. Technical Report TR 99-034, Department of Computer Science, University of Minnesota.

[Karypis et al., 1999a] Karypis, G., Aggarwal, R., Kumar, V., and Shekhar, S. (1999a). Multilevel hypergraph partitioning: Application in vlsi domain. *IEEE Transactions on VLSI Systems*, 20(1). A short version appears in the proceedings of DAC 1997.

[Karypis et al., 1999b] Karypis, G., Han, E., and Kumar, V. (1999b). Chameleon: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer*, 32(8):68–75.

[Karypis and Kumar, 1995] Karypis, G. and Kumar, V. (1995). Analysis of multilevel graph partitioning. In *Proceedings of Supercomputing*. Also available on WWW at URL http://www.cs.umn.edu/˜karypis.

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

[Karypis and Kumar, 1998b] Karypis, G. and Kumar, V. (1998b). METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Department of Computer Science, University of Minnesota. Available on the WWW at URL *http://www.cs.umn.edu/˜metis*.

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

[Karypis and Kumar, 1999a] Karypis, G. and Kumar, V. (1999a). A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1). Also available on WWW at URL http://www.cs.umn.edu/˜karypis. A short version appears in Intl. Conf. on Parallel Processing 1995.

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

[Karypis and Kumar, 2000] Karypis, G. and Kumar, V. (2000). Multilevel k-way hypergraph partitioning. *VLSI Design*.

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

[Krishnamurthy, 1984] Krishnamurthy, B. (May 1984). An improved min-cut algorithm for partitioning VLSI networks. *IEEE Transactions on Computers*, Vol. C-33.

[Lengauer, 1990] Lengauer, T. (1990). *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, New York, NY.

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

[Pothen et al., 1990] Pothen, A., Simon, H. D., and Liou, K.-P. (1990). Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis and Applications*, 11(3):430–452.

[Sanchis, 1989] Sanchis, L. A. (1989). Multiple-way network partitioning. *IEEE Transactions on Computers,* pages 62–81.

[Schloegel et al., 1999] Schloegel, K., Karypis, G., and Kumar, V. (September 1999). A new algorithm for multi-objective graph partitioning. In *Proceedings of Europar 1999*.

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

[Simon and Teng, 1993] Simon, H. D. and Teng, S.-H. (1993). How good is recursive bisection? Technical Report RNR-93-012, NAS Systems Division, NASA, Moffet Field, CA.

[Walshaw et al., 1997] Walshaw, C., Cross, M., and Everett, M. G. (1997). Dynamic load-balancing for parallel adaptive unstructured meshes. *Parallel Processing for Scientific Computing*.