An Algebraic Multigrid Based Algorithm for Bisectioning General Graphs

Dorit Ron* Sharon Wishko-Stern Achi Brandt

Technical Report MCS05-01
January 2005

Department of Computer Science and Applied Mathematics,
Weizmann Institute of Science, Rehovot 76100, Israel

Keywords: *graphs, bisectioning, algebraic multigrid (AMG), coarsening, weighted aggregation, interpolation, relaxation, simulated annealing, lowest common configuration (LCC)*

*Correspondence should be addressed to D. Ron, Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel, Tel: +972-8-9342141, Fax: +972-8-9342945, E-mail: dorit.ron@weizmann.ac.il

Abstract

The Graph partitioning problem is widely used and studied in many practical and theoretical applications. The problem concerns the partitioning of the nodes of a graph into a given number of disjoint subsets, while minimizing the number of inter-edges, edges that connect one set to the other. Furthermore, the sets must be of roughly equal size. In this work we present a new multilevel algorithm for partitioning graphs into two disjoint sets. The algorithm is inspired by the Algebraic Multigrid approach which is based on weighted edge contraction rather than simple contraction. The algorithm uses an extended energy functional that reflects the trade off between the constraint regarding the size of the sets, and the minimization of the number of inter-edges. It allows a variable amount of imbalance at coarser levels, which is reduced gradually as the uncoarsening proceeds. Experimental tests show that (except for a special class of graphs in which there exists a subgraph which is strongly connected within itself) the minimum cost obtained using our algorithm is comparable to, and in some cases even better than the best known results. Furthermore, the averages and, in particular, the standard deviations are lower than those achieved by hMETIS (the most popular up-to-date partitioning algorithm), i.e., our algorithm is less sensitive to the choice of sequences of random numbers.

1 Introduction

In the most general form, the *graph partitioning problem* concerns the partitioning of a graph into a given number of disjoint subsets of nodes of roughly equal size, while minimizing a given objective functional that reflects the amount of connection between the subsets. The problem comes up in many fields connected to computer science including sparse matrix-vector multiplication, image processing [40] and parallel computation. It is also central in many subfields connected to Very Large Scale Integration (VLSI) of circuits and systems [14]. The problem is known to be NP-hard [19]. A special instance of the graph partitioning problem, in which the graph must be divided into two approximately equal sized subsets, is known as the graph *bisectioning* problem. Although the bisectioning problem seems an easier problem, it is also known to be NP-hard [20]. An algorithm is said to approximate a minimization problem within a ratio f ($f \geq 1$) if it runs in polynomial time and outputs a solution whose objective functional is at most f times the minimal one. The best known approximation algorithm for bisectioning a general graph (consisting of n nodes, n even) achieves a ratio of $O(\log^2 n)$ from the minimum [17]. Therefore, all known practical algorithms for partitioning are heuristics that merely return approximations to

the optimal bisection.

The problem has attracted a lot of attention, and has given rise to several heuristic solvers. Different heuristics that address the problem are surveyed in [1]. Those different heuristics solvers may be divided into four major classes: *Move based algorithms*, that iteratively produce a new candidate solution based on a previous feasible solution [31, 18, 24, 21, 12]; *Geometric algorithms* that use the graph's coordinates information [13, 22, 34]; *Structural algorithms* that operate on the graph's combinatorial structure [41, 15, 35]; and *Multilevel algorithms* [3, 23, 11, 29, 30, 47, 42, 46, 2].

Since 1995, multilevel algorithms have become the most popular approach. This class of graph partitioning algorithms recursively reduces the size of the graph (i.e., coarsens the graph) by collapsing vertices and edges, partitions the smallest (coarsest) one and then uncoarsens it to construct partitions for all subsequent larger graphs until a partition for the original graph is obtained. The coarsening process is based on matching nodes according to some heuristics (e.g., heavy edge matching [29]). The matched nodes are "collapsed" (merged) to create a new *multinode* with a volume equal to the sum of their volumes. Similarly the edges connecting multinodes are the sum of the edges incident on the corresponding matched nodes. The coarsest level (smallest graph) is then partitioned by various heuristics. During the uncoarsening (interpolation) process the matched nodes are simply separated and each gets the partition of its multinode. To get better results different refinement processes which are modifications of the well known Kernighan-Lin [31] and Fiduccia-Mattheyses [18] move based algorithms are used [11, 23, 29, 30, 47, 42, 46].

Different ways of matching nodes were proposed and tried by Karypis and Kumar in [27] and [29]. A greedy heuristic, called *heavy edge matching*, in which edges with higher weight are more likely to be selected proved itself (experimentally) to be superior to other heuristics and was later used in [29, 30, 47, 42, 46, 2]. Based on this work they have developed a software package for partitioning large graphs named METIS. This package was later extended to the partitioning software package for large hypergraphs named *hypergraph*-METIS (hMETIS) [27], by implementing the algorithms described in [30, 25]. By presenting a graph as a hypergraph, were all hyperedges are of size two, hMETIS can also be used for partitioning graphs. In fact, it actually performs better than METIS, especially as it allows many partitioning trials, each resulting with a different partition due to stochastic elements in the algorithm.

Another commonly used partitioning software package, JOSTLE, has been developed by Walshaw et al. at Univ. of Greenwich. It includes implementation of the greedy [15],

recursive coordinate bisection [41], multilevel Kernighan-Lin [47] and iterated multilevel Kernighan-Lin [46] algorithms. The package also includes an implementation of a combined evolutionary/multilevel algorithm [42]. Experiments showed that the combined evolutionary/multilevel, although not a practical method for applications in which partition must be found rapidly, is very successful at computing very high quality partitions compared to METIS. The results were not compared to those of hMETIS since the definitions of the imbalance factor are incompatible.

This report concentrates on the bisectioning problem of sparse graphs. It suggests a new multilevel bisectioning algorithm which is inspired by the Algebraic MultiGrid (AMG) scheme [7, 8, 4, 37, 43, 44]. General multilevel techniques have been successfully applied to various areas of science (e.g. physics, chemistry, engineering, etc.) [6, 9]. AMG-type algorithms were originally developed for solving linear systems of equations resulting from the discretization of partial differential equations. Lately they have been applied to various other fields, yielding for example novel methods for image segmentation [39] and for the graph linear arrangement problem [38]. In the context of graphs it is the Laplacian matrix that represents the related set of equations attached to a graph. The main difference between the AMG approach and other multilevel ones is the coarsening scheme. While the latter may be viewed as *strict* aggregation process, the AMG coarsening is actually a *weighted* aggregation: each node may be divided into *fractions*, and different fractions belong to different aggregates (multinodes). In both cases, these aggregates will form the nodes of the *coarser level*. As AMG solvers have shown, *weighted*, instead of *strict* aggregation is important in order to express the *likelihood* of nodes to belong together; these likelihoods will then accumulate at the coarser levels of the process, automatically reinforcing each other where appropriate. This enables more freedom in solving the coarser levels and avoids making hardened local decisions, such as edge contractions, before accumulating the relevant global information, while a strict aggregation may lead to inconsistency between local and global considerations.

The disaggregation consists of projecting to a finer level the final partition obtained at the next coarser level. This initial fine level partition is being further improved by applying local processing first of strict minimization followed by Simulated Annealing (SA) [32]. SA is a general method used to escape local minima. By introducing a temperature like parameter, moves which increase the cost function one wants to minimize are accepted with some non-vanishing probability. These algorithms are usually extremely inefficient, since they require exponential slow temperature decrease to approach the true minimum. In the multilevel framework, however, SA is aimed at searching only for *local* changes, with rapid cooling at each level, which guarantees the preservation of large-scale solution

features inherited from coarser levels.

To demonstrate the importance of our weighted coarsening scheme, we have constructed a family of graphs in which some strong local connections (edges) would probably push any heuristic solver (which is based on strict clustering) to find a local minimum away from the global one. We have tested these examples with hMETIS and with our new algorithm. It turned out that while hMETIS hardly ever finds the global minimum (even with many hundreds of trials), our algorithm uniformly converged to the global minimum and was not trapped in any of the local ones. In addition, we have tried our algorithm on many other graphs taken form [45]. We compare our results with the best partitions found to date which appear in Walshaw's web site [45] as well as to the hMETIS results. Experimental tests show that the minimum cost obtained using our algorithm is in most cases practically equal to, and in some cases even better than the best known results. Furthermore, the averages and, in particular, the standard deviations are lower than those achieved by hMETIS, i.e., our algorithm is less sensitive to the choice of sequences of random numbers.

An exception where our present algorithm performs more poorly is a special class of graphs which contains subgraphs strongly connected within themselves. A correction for the algorithm to deal with such cases is discussed below but has not yet been implemented.

The present report is divided as follows: The problem definition is given in section 2. Our algorithm is described in section 3. Finally, in sections 4 and 5 we report on the algorithm general performance, draw conclusions and suggest future work.

2 Problem Definition

Given a weighted graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is a set of n *vertices* and E is a set of *edges*, denote by w_{ij} the non negative weight assigned to an edge ij, $i, j \in V$ (if there is no edge ij then $w_{ij} = 0$) and by v_i the non negative volume that may be assigned to vertex i (if no volume is assigned consider all nodes to have an equal volume). The purpose of the *k-way graph partitioning* problem is to divide V into a (usually small) number of disjoint subsets $R_1, R_2, \ldots, R_k$, called *regions*, each with total volume between two given bounds v^{min} and v^{max}, so that the total weight of all inter-region edges (*cutsize*, *partition-cost*) is minimal. Or, more formally: Given a weighted graph $G = (V, E)$, a positive integer k and two bounds v^{min} and v^{max}, find k subsets $R_1, R_2, \ldots, R_k$ of V such that

1. $\cup_{i=1}^{k} R_i = V$ and $R_i \cap R_j = \emptyset$, $\forall (i \neq j)$, $i, j \in \{1, 2, \ldots k\}$.

2. $v^{min} \leq \sum_{i \in R_j} v_i \leq v^{max}, i \leq |V|, j = \{1, 2, \ldots k\}$

3. $\sum_{\substack{i \in R_p, j \in R_q, \\ p \neq q}} w_{ij}$ is minimized among all partitions of V that satisfy 1 and 2.

Bisectioning or *bipartitioning* is the problem for $k = 2$, where usually the bounds on the two volumes are tight to obtain nearly balanced bisection. In particular, the maximum imbalance allowed (in percentage), denoted by *imbalance_factor* (usually given by the user), determines v^{min} and v^{max} :

$$v^{min} = \frac{(50 - imbalance\_factor)}{100} * \sum_{i \in V} v_i \tag{1}$$

$$v^{max} = \frac{(50 + imbalance\_factor)}{100} * \sum_{i \in V} v_i \quad . \tag{2}$$

For other major variant formulations of the partitioning problem see [1]. Although the bisectioning problem seems an easier problem than the general partitioning problem, it is also known to be NP-hard [20]. The methods developed below for bisectioning have natural extensions to general partitioning.

3 The Multilevel algorithm for bisectioning

In this section we introduce the new multilevel algorithm for the graph bisectioning problem. The heuristic algorithm works in time linear in the number of vertices for sparse graphs. By sparse we mean graphs for which the number of edges, $|E|$, is $O(|V|)$ and hence the average degree (denoted by avg_deg) is much smaller than the number of vertices, i.e., $avg\_deg \ll |V|$. The algorithm is based on the Algebraic MultiGrid (AMG) scheme.

Formally, a multilevel graph bisectioning algorithm consists of the following three stages:
Coarsening - Given a (weighted) graph $G_0 = (V_0, E_0)$, a hierarchy of decreasing size graphs $G_0, G_1, \cdots, G_k$ is derived from it.
Bisectioning - A bisection of G_k, denoted by B_k, is computed.
Uncoarsening (disaggregation) - The bisection of G_k, B_k, is projected to create an approximate bisection of G_{k-1}, which is then refined to create B_{k-1}. This process is continued until a bisection of the given graph G_0 is obtained.

As in the general AMG setting, the choice of the coarse variables (aggregates), the derivation of the coarse problem which approximates the fine one and the design of the

coarse-to-fine disaggregation (uncoarsening) process are all determined automatically as described below. For a schematic recursive definition of the algorithm see the box Algorithm Bisection. For details see the following subsections.

Algorithm Bisection$(G_i = (V_i, E_i))$
If $|V_i| \leq 20$
  Solve the problem directly.
Else
  Construct the set V_{i+1} of nodes of G_{i+1}, each being a weighted aggregate of nodes in V_i
  Construct the set E_{i+1} of edges of G_{i+1} by weighted summation of edges in E_i
  Bisection$(G_{i+1} = (V_{i+1}, E_{i+1}))$
  Disaggregate the bisection of G_{i+1} to create an initial bisection of G_i
  Improve the bisection of G_i by local processing
Return the bisection of G_i
end.

3.1 Preliminaries and notations

Let $G_0 = (V_0, E_0)$ be the given graph. During the coarsening phase of the multiscale process the current level graph $G_i = (V_i, E_i)$ is coarsened to create the next level graph - $G_{i+1} = (V_{i+1}, E_{i+1})$. In the description of the algorithm we occasionally refer to the current level (fine) graph as $G_f = (V_f, E_f)$ and to the next level (coarse) graph as $G_c = (V_c, E_c)$. The nodes (vertices) that will serve as the *seeds* of the next level nodes are selected from the current set of nodes, V_i, and gathered into a new set denoted by $C \subset V_i$. We denote the complement of this set by $F = V_i \setminus C$. At the end of the coarsening phase we get a coarsened graph $G_{i+1} = (V_{i+1}, E_{i+1})$, where $|V_{i+1}| = |C|$ and E_{i+1} is a new set of edges. Note that the nodes in V_{i+1} have generally different ordinal numbers than their seeds in V_i. Each vertex i is assigned a *volume*, denoted by v_i. (In most applications all nodes of G_0 are assigned an equal volume, 1.) In the uncoarsening phase, each node is assigned to a region $R \in \{R_1, R_2\}$. We denote the complement of R by $\bar{R}$, i.e., $\bar{R}_1 = R_2$, $\bar{R}_2 = R_1$.

We further generalize the problem as follows. In order to get a higher bisection quality we allow variable amount of imbalance which is gradually reduced as the uncoarsening phase proceeds. In other words the balance constraint is looser at coarser levels: Initially, on G_0, we use the $imbalance\_factor$ given by the user (see equations 1 and 2). At every subsequent level we define a larger $imbalance\_factor$. We denote the imbalance factor used

at the i-th level by $imbalance\_factor_i$. More details follow in Section 3.2.3.

When solving the coarsest level and during the disaggregation phase, two objectives should be considered: Minimizing the *cutsize*, i.e., $\min \sum_{i \in R, j \in \bar{R}} w_{ij}$, while satisfying the balance constraint. Note that those objectives might conflict. We therefore introduce a new *energy functional* En which has to be minimized and which grows exponentially when the balance constraint is violated. As a result, while a feasible bisection with higher cutsize is usually preferable over an infeasible bisection with lower cutsize, an infeasible bisection that only slightly violates the volume constraints will be chosen over a bisection that satisfies the balance constraint but has much higher cutsize. Let G_i be a given graph with a given bisection where $imbalance\_factor_i$ is the percentage by which imbalance is allowed. Let $current\_imbalance$ be the current bisection imbalance defined as $50 * |\sum_{i \in R} v_i - \sum_{i \in \bar{R}} v_i| / |V_0|$. $En(G_i)$ is defined by

$$En(G_i) \stackrel{def}{=} cutsize \cdot exp\left(\rho \cdot max\left[(current\_imbalance - imbalance\_factor_i), 0\right]\right) \qquad (3)$$

The value of ρ is determined as follows: At the coarsest level in order to determine an initial bisection, and at any other level to allow a good initial assignment of nodes at disaggregation (see below Section 3.4.1), ρ is chosen so that if the deviation from the $imbalance\_factor_i$ equals half of the heaviest node's relative volume $v_{max}/|V_0|$, the energy is larger than the cutsize by $initial\_punish\_cost$. In this case ρ is the solution of the following equation:

$$1 + initial\_punish\_cost = exp\left(\rho \cdot \frac{0.5 \cdot v_{max}}{|V_0|} \cdot 100\right) ,$$

where $initial\_punish\_cost$ was set to 0.10. At each level, after the initial partitioning has been obtained, the value of ρ is updated. ρ is chosen so that when the deviation from the $imbalance\_factor$ equals half the relative volume of the heaviest node along the cutsize, $v_{max\_border}/|V_0|$, the energy is larger than the cutsize by $border\_punish\_cost$. In this case ρ is the solution of the following equation:

$$1 + border\_punish\_cost = exp\left(\rho \cdot \frac{0.5 \cdot v_{max\_border}}{|V_0|} \cdot 100\right) ,$$

where $border\_punish\_cost$ was set to 0.02.

We finally introduce one more notation. Let $w_S(ij)$ denote the normalized weight of an edge ij with respect to the set of nodes S and to the vertex i, defined by

$$w_S(ij) = \frac{w_{ij}}{\sum_{k \in S} w_{ik}} \quad .$$

A list of all the parameters used in the algorithm is given in the Appendix.

3.2 Coarsening

The construction of a coarse graph from a given fine one is divided into three stages: first a subset of the fine nodes is chosen to serve as the seeds of the aggregates (the nodes of the coarse graph), then the rules of aggregation are determined, thereby establishing the fractions of each non-seed node belonging to each aggregate, and finally the strength of the connections (or edges) between the coarse nodes is calculated.

3.2.1 Coarse seed selection

The algebraic representation of a graph $G(V, E)$ is given by the *graph Laplacian* A, a $|V| \times |V|$ matrix whose terms are defined by

$$a_{ij} \stackrel{def}{=} \begin{cases} -\sum_{k \neq i} w_{ik} & \text{for } (i = j) \\ w_{ij} & \text{for } i \neq j \ , \ ij \in E \\ 0 & \text{otherwise} \end{cases} .$$

The construction of the coarse level graph starts with a construction of the subset C of the fine level set of nodes V (recall that the complement of C is F). On one hand, this set of seeds should be representative of the given fine graph so that each F-node is "strongly connected" to C, which should guarantee good uncoarsening. On the other hand it should obey some balancing rules to maintain as even a distribution of the total graph volume as possible. The set C should also include nodes with exceptionally large volume, or nodes that, if used as seeds, would tend to aggregate around them an exceptionally large volume of F-nodes (a rule that will be modified in special situations; see Section 4.3).

Define the *future-volume*, ϑ, of a fine-level node i as

$$\vartheta_i \stackrel{def}{=} v_i + \sum_{j \in F} v_j \cdot w_V(ji) \ ,$$

which is a measure of how large an aggregate seeded by i might grow. We start with an empty set C, hence $F = V$, and then sequentially transfer nodes from F to C, employing the following steps. Nodes with future-volume larger than η times the average of ϑ are automatically added to C; we used $\eta = 2$. Next, a criterion for further increasing C, and a practical method to control its quality can be based on sweeps of *compatible relaxation* [5]. We use a compatible *Gauss-Seidel* (GS) relaxation. A vector x defined on V is relaxed by

GS relaxation that avoids relaxing those variables of x that belong to nodes in C. More precisely, consider a relaxation process for the system of equations $Ax = 0$ initialized by

$$x_i = \begin{cases} 0 & \text{if } i \in C \\ 1 & \text{if } i \in F \ . \end{cases}$$

In each GS *sweep*, the equations are scanned in their natural order, and each equation $i \in F$ in its turn is satisfied by replacing the current approximation to the associated unknown x_i, with the new value $\tilde{x}_i = a_{ii}^{-1}(-\sum_{j \neq i} a_{ij} x_j)$. The convergence rate of each x_i is then checked. The set C is guaranteed to be good when (and only to the extent that) the compatible relaxation exhibits uniformly fast convergence rates (of all x_i to 0). When these rates are too slow, a diluted subset of the slow-to-converge F-nodes should be added to C. In particular, we divide the current set F into *num_groups* groups (we set the value of *num_groups* to three) of decreasing convergence slowness. Each group is then sorted in ascending order of the ϑ_is of the related F-nodes. We then scan the groups' nodes starting with the slowest to converge group and add nodes to C (starting with those having the largest ϑ_i). When a node is added to C the node's neighbors are marked so that they will not be added to C at the current iteration. In order to reduce the variability of the volumes of the coarse nodes, nodes with large ϑ and a relative weak connection to C (i.e., lower from a given threshold Q) are chosen with higher priority. We used $Q = 0.4$. This process is repeated iteratively until C is large enough, i.e., as long as $\frac{|C|}{|V|} < \frac{1}{2}$. In each iteration, ν GS sweeps are done, where $\nu = 8$. For complete details see Procedure Fix_C_Points.

Remark- For convenience we are currently using a library $O(n \cdot log(n))$ sorting algorithm. However, since only a very rough ordering is really needed, this can be replaced by a rough bucketing sort which has $O(n)$ complexity. This remark applies as well for all cases below where we have used an exact sort.

The chosen set C of nodes is the set of seeds of the aggregates, which will become the coarse level nodes. It is left to determine the aggregation such that the coarse graph bisectioning will be a good approximation to the bisectioning of the finer graph and then to construct all coarse edges in a way that will encapsulate the connectivity information of the finer graph. We first introduce the notion of aggregation weights.

3.2.2 Aggregation weights

The set C of coarse variables is chosen so that, once a partition of the coarse nodes (aggregates) is given, a partition of all fine nodes will quite naturally follow. The *aggregation*

Procedure Fix_C_Points(Q, η, ν)

$C \leftarrow \emptyset$, $F \leftarrow V$
Calculate ϑ_i for each $i \in F$
$C \leftarrow$ nodes with $\vartheta > \eta \cdot$(average of ϑ)
$F \leftarrow V \setminus C$
Recalculate ϑ_i for each $i \in F$
While $|C|/|V| < \frac{1}{2}$
  Set $x_i = 0$ $\forall i \in C$ and $x_i = 1$ $\forall i \in F$
  Relax $Ax = 0$ by ν GS relaxation sweeps on F
  Sort x in decreasing order (larger x indicates slower convergence)
  Divide the sorted x into equal sized *num_groups* groups
  Sort each group in decreasing order of the ϑ_i of the related F-nodes
  Unmark all nodes in F
  For each group g (starting with the slowest to converge)
    For each unmark i in g
      If $(\sum_{j \in C} w_{ij} / \sum_{j \in V} w_{ij}) \leq Q$
        Move i from F to C
        Mark the neighbors of i
  Recalculate ϑ_i for each $i \in F$
Return C
End.

weights are defined to be a sequence of fractions P_{ij} (see below), where i is a fine node, and $j \in C$ is the seed of one of the aggregates to which the fraction P_{ij} of i "belongs". That is, P_{ij} represents the fraction of i that tends to belong to the same region as j.

In classical AMG, these aggregation (interpolation) weights are based on the "strength of couplings", a_{ij} of A. The following are formulae for such generalized couplings. (Note that, for every i, $\sum_j a_{ij} = 0$ while the normalized couplings, defined below, satisfy $\sum_j a_{ij}^{(\nu)} = 1$.) The *$\nu$-generation normalized couplings* $a_{ij}^{(\nu)}$ are recursively defined by

$$a_{ij}^{(\nu)} \;=\; \begin{cases} 1 & \text{for} \quad i \in C \;,\; j = i \\ 0 & \text{for} \quad i \in C \;,\; j \neq i \\ 0 & \text{for} \quad i \notin C \;,\; j = i \\ \widetilde{a}_{ij}^{(\nu)} & \text{for} \quad i \notin C \;,\; j \neq i \;, \end{cases} \tag{4}$$

where

$$\tilde{a}_{ij}^{(\nu)} = \begin{cases} -a_{ij}/a_{ii} & : \nu = 1 \\ \sum_l a_{il}^{(\nu-1)} a_{lj}^{(\nu-1)} / (1 - \sum_l a_{il}^{(\nu-1)} a_{li}^{(\nu-1)}) & : \nu > 1 \ . \end{cases} \tag{5}$$

The purpose of the ν-generation normalized couplings calculation is to consider not only direct couplings (edges) between a node i to $j \in C$, but also to accumulate indirect couplings in which $j \in C$ is reached via one or more (depending on ν) F-nodes. This calculation reveals to which of the C-nodes a particular i is mostly connected, thus enables a more accurate and reliable calculation of the couplings of the next coarse level (see below). Clearly, for $i \in C$ and for any ν, $a_{ii}^{(\nu)} = 1$ and thus it is necessary to actually calculate new couplings only for $i \notin C$.

The 1-generation normalized couplings is merely a normalization of the original couplings which indeed yields for every i, $\sum_j^{(1)} a_{ij} = 1$. This normalization breaks the original symmetry of the couplings, i.e., while $a_{ij} = a_{ji}$, $a_{ij}^{(1)}$ is usually different from $a_{ji}^{(1)}$. Consider the graph sample given in Figure 1(a), where nodes 2, 5 and 8 are C-nodes (indicated by double circles) and a'_{ij} is a short notation for $a_{ij}^{(1)}$. We will concentrate, for example, on the calculation of the 2-generation normalized couplings of node 1. Clearly its 1-generation couplings sum up to 1, i.e., $a'_{12} + a'_{13} + a'_{16} + a'_{17} = 1$, and so do the corresponding sums of all other nodes. For the 2-generation couplings calculation we replace every coupling adjacent on 1 which *does not lead* to a C-node, i.e., replace a'_{13}, a'_{16} and a'_{17}. For example, node 6 is connected to nodes 1, 5, 8 and 7, thus a'_{16} is replaced by $a'_{16}a'_{61}$, $a'_{16}a'_{65}$, $a'_{16}a'_{67}$ and by $a'_{16}a'_{68}$, that is, node 1 is now connected to 1, 5, 7 and 8, via those four couplings. Note that the sum of these new four couplings still equals a'_{16}, and that node 1 is now connected not only to one C-node 2 as before, but also to 5 and 8. We similarly replace a'_{13} and a'_{17}. The resulting couplings of node 1 are presented in Figure 1(b). Their sum still equals 1, however, a self loop (i.e., a coupling from 1 to itself) is included. To normalize all *other* couplings one should divide each of them by one minus the strength of the self loop. Thus for example, the final coupling between node 1 and 5 is

$$\frac{a'_{13}a'_{35} + a'_{16}a'_{65}}{1 - (a'_{13}a'_{31} + a'_{16}a'_{61} + a'_{17}a'_{71})} \ .$$

The self loop is set to zero, and node 1 is finally connected to three C-nodes: 2, 5 and 8.

We have used $\nu = 2$ followed by a procedure called *$\tilde{\nu}$-selective-generation* in which the $\tilde{\nu}$-generation (in our application $\tilde{\nu} = 3$) is applied only partially, i.e., only to a small subset of the fine nodes, those which have particularly "weak" connections to C, e.g. smaller than 0.3.

To curb complexity, the number of fractions of each node should be restricted by a proper

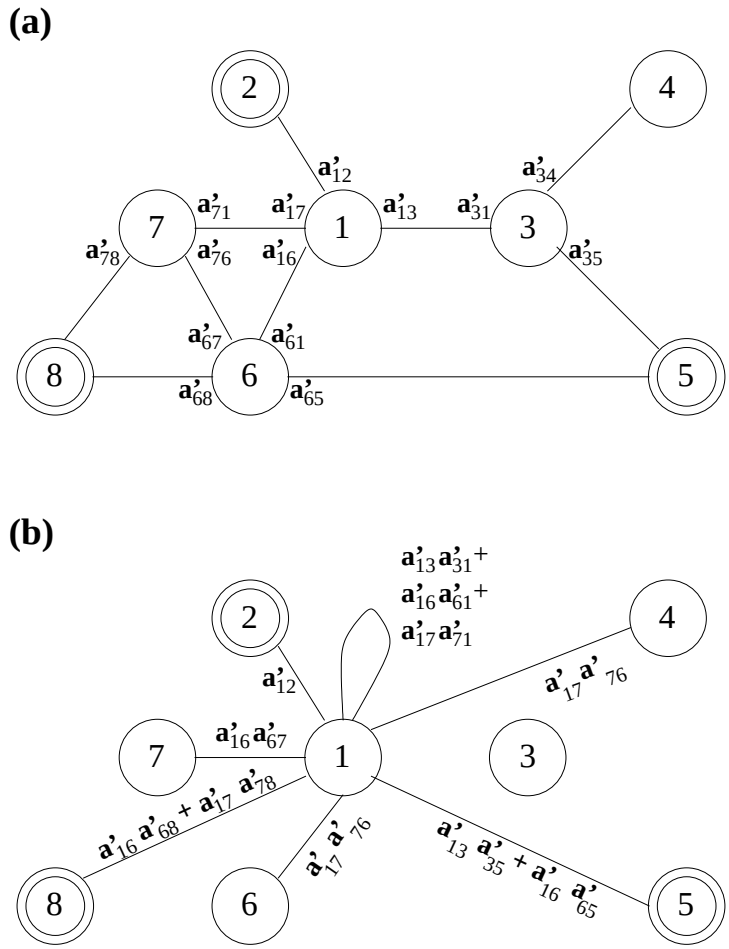

Figure 1: A graph sample. C-nodes are double circled. (a) The original graph, where the a'_{ij}s denote the 1-generation normalized couplings between i and j. (b) The resulting *non-normalized* 2-generation couplings of node 1.

choice of the thresholds α_i defined below. Denote by $\hat{P}_i$ the set of the $\bar{\nu}$ (the final generation used for node i: either ν or $\widetilde{\nu}$) generation normalized couplings: $\hat{P}_i=\{a_{ij}^{(\bar{\nu})}|j \in C\}$, and define for each $i \notin C$ a *coarse neighborhood* $N_i = \{j \in C|a_{ij}^{(\bar{\nu})} \geq \alpha_i\}$, where α_i is determined by the demand that the size $|N_i|$ should not be too large. To compute the value of α_i we first sort $\hat{P}_i$ by decreasing order and define α_i to be

$$\alpha_i = \begin{cases} 0 & \text{if } |\{a_{ij}^{(\bar{\nu})} > 0 | j \in C\}| \leq r \\ \hat{P}_i[r] & \text{otherwise} \end{cases} ,$$

where the *interpolation order* parameter, r, is initialized by $r_0 = 6$ at the finest level and increased as a function of level L to $r = r_0 + R$, where $R \stackrel{def}{=} \log(max(1, |E_0|/|E_L|))$.

Finally, we define the aggregation weights matrix P (of size $|V| \times |C|$) by

$$P_{ij} = \begin{cases} a_{ij}^{(\bar{\nu})} / \sum_{k \in N_i} a_{ik}^{(\bar{\nu})} & \text{for } i \in F,\ j \in N_i \\ 1 & \text{for } i \in C,\ j = i \\ 0 & \text{otherwise} \end{cases} .$$

Note that for every i, the aggregation weights satisfy $\sum_{j \in N_i} P_{ij} = 1$.

3.2.3 The coarse problem

Let $I(k)$ be the ordinal number of the coarse graph node that represents the aggregate around a seed whose ordinal number at the fine level is k. A coarse edge between two coarse nodes should represent the total weight of all the fine-level edges (connections) between the corresponding aggregates. Thus, we followed the *weighted aggregation* scheme used in [39, 38]. An edge connecting two *coarse* nodes $p = I(i)$ and $q = I(j)$ is assigned with the weight

$$w_{pq}^c = \sum_{k \neq l,\ k,l \in V_f} P_{ki} w_{kl} P_{lj} .$$

Dilution - Note that to control the running time of the algorithm, it is often important to decrease the total number of edges of the constructed graph. This is achieved by using the following two parameters:

- The maximum allowed coarse neighborhood size r which restricts the number of aggregation weights allowed for each vertex $i \in F$ (see Section 3.2.2).
- Edge filtering threshold, ϵ. Every *relatively* weak edge ij is deleted from the created coarse graph; namely, ij is deleted if both $w_{ij} < \epsilon \cdot s_i$ and $w_{ij} < \epsilon \cdot s_j$, where $s_i = \sum_k w_{ik}$. We used $\epsilon = 0.001$.

The volume of a coarse node $i = I(k)$, $v^c_{I(k)}$, should reflect the volumes of the nodes in the aggregate around k including k itself. Thus, we define its volume to be $v^c_{I(k)} \stackrel{def}{=} \sum_{j \in V_f} v_j P_{jk}$, i.e., it equals to the previous volume of its seed k plus the relative parts of the volumes of other fine nodes it represents. Note that during the coarsening process the total volume of all vertices is conserved, i.e., for all coarse graphs, $G_i = (V_i, E_i)$, obtained in the coarsening phase $\sum_{u \in V_0} v_u = \sum_{u \in V_i} v_u$. Also, under the assumption that all nodes in the original graph have volume one $|V_0| = \sum_{u \in V_0} v_u$.

The constructed coarse problem is again a graph bisection problem with v^{min} and v^{max} constraints. As explained, during the coarsening process the number of vertices in the graph decreases while their total volume is conserved. Thus, the average volume of a node i, v_i, increases. We wish to enable the transfer of nodes from one side of the partition to the other without violating the constraints. Therefore, the $imbalance\_factor$ (provided by the user) should increase with respect to the graph level, i.e., the constraints should be looser when the graph level is higher. In each level we chose the $imbalance\_factor$ to be equal to the relative volume of the heaviest node. That is, the imbalance factor used at the i'th level denoted by $imbalance\_factor_i$ has the value

$$imbalance\_factor_i = \max\left(imbalance\_factor, \frac{v_{j(i)}}{|V_0|}\right) ,$$

where $j(i)$, is the heaviest node at level i, i.e., it has the maximum volume.

3.3 Coarsest level solution

Solving the coarsest level, which consists of no more than 20 nodes (otherwise a coarser level should be introduced for efficiency) is preformed directly by simply trying all possible bisections. Note that for a graph that consists of 20 nodes there are just 524287 possible bisections (not all necessarily satisfy the balance constraint). Our algorithm will usually accept for further processing *several* bisections, provided that they all have relatively low values of En and they are all substantially *different* from each other. More precisely, consider first the best obtained bisection, i.e., the one that has the minimal En. Scanning other bisections, according to their increasing En, we pick additional ones if the total volume of the nodes at which they differ from the ones already picked, is more than a certain fraction, ω, of the total vertices volume (we have used $\omega = 0.08$). For example, let s^0 be the optimum bisection of G_i and s^0_j be the bisection region (R_1 or R_2) assigned to

node $j \in V_i$. A bisection s^k is considered to be sufficiently different from s^0 if

$$\min \Big(\sum_{j \in V_i,\ s_j^0 \neq s_j^k} v_j \ , \sum_{j \in V_i,\ \bar{s}_j^0 \neq s_j^k} v_j \Big) > \omega \cdot |V_0| \ . \tag{6}$$

The maximum number of bisections chosen from the coarsest level is denoted by *num_coarse_solutions* (although we have fixed it to 40, a perhaps better way would be to choose it proportional to the original graph size). See Section 3.4.5 for the subsequent selections that reduce this number at finer levels.

3.4 Disaggregation

Having solved the coarse problem, an approximate bisection to the original (or the-next-finer-level) problem is obtained by *disaggregation*, the analog of the coarse-to-fine multigrid interpolation. The disaggregation is done in two phases. The *initialization* of the fine level bisection, which is subsequently improved by several *relaxation* sweeps, first compatible then regular, with or without stochastic elements, as explained below.

3.4.1 Initialization by layers

Define $V' \subset V$ to be the subset of nodes that have already been assigned to one of the regions. Initially $V' = \emptyset$. At first, having obtained a bisection of the coarse level aggregates, each node i for which its aggregation weights $\sum_{j \in R_k} P_{ij}$ is almost 1 (≥ 0.95) for either $k = 1$ or $k = 2$, is assigned to that region R_k. Clearly all seeds are bisectioned and added to V'. Then iteratively assign each fine node $i \in V \setminus V'$ to the region R_k if $\sum_{j \in R_k} w_V(ij) \geq certainty$[1]. The value of *certainty* is initialized with 0.95. It is decreased by 0.05 when the number of nodes that were assigned in the last iteration is smaller than $max(10, 0.1 \cdot \text{graph size})$ and stops when its value is less than 0.90. Thus, the sequence in which the nodes are placed is roughly in decreasing order of their relative connection to V'. Finally assign a partition to the rest of the nodes using energy considerations. All fine nodes that have not yet been assigned concentrate along the cut (between R_1 and R_2). We assign each fine node $i \in V \setminus V'$ to the region R_k so that the value En of the bisection, at that moment when i is being placed, is minimized. Note that some of the neighbors of i may not have yet been

[1] In the results presented below we have actually used a slightly different criterion: $\sum_{j \in R_k} w_{V'}(ij) \geq certainty$, which proves to perform well.

assigned to a region. More precisely, $i \in V \setminus V'$ is assigned to a region R_k $(k = 1, 2)$ if

$$\sum_{j \in R_k} w_{ij} \geq \sum_{j \in V \setminus V'} w_{ij} + \sum_{j \in V', j \notin R_k} w_{ij} \ ,$$

until no new assignments were made. The rest of the nodes $i \in V \setminus V'$ are then assigned by majority: i is assigned to the region R_k if

$$\sum_{j \in R_k} w_{ij} > \sum_{j \in V', j \notin R_k} w_{ij} \ .$$

If the terms in the inequality above are actually equal, then it will be assigned to either R_k with probability 0.5.

3.4.2 Strict minimization

The simple disaggregation (initialization by layers) is not likely to be accurate enough. It should therefore be followed by several sweeps of *compatible* relaxation, motivated in [5], which is like the minimization described below, but avoids changing the positions of the seeds (C-nodes). This will produce much improved disaggregation still compatible with the coarse bisection before changing it by *regular* relaxation.

A strict node by node minimization (regular relaxation) is simply the process of assigning each node to R_1 or R_2 according to which of the two has a lower value of En. The minimization is applied along the cut line. Given a node $i \in R$ that is located on the cut, i.e., there exists some edge ij such that $j \in \bar{R}$, the new value of En associated with the assignment of i to region $\bar{R}$ is calculated. If the new value of En is lower or equal to the value of En before the change, then the change is accepted. This process should be applied to all nodes on the cut and continue for sufficiently many times. We run it for maximum of *num_minimization* times chosen to be 10, as long as the energy is further reduced. Note that the calculation of the new En is done locally since the cutsize *change* when node k is transferred from R to $\bar{R}$ is given by

$$\sum_{j \in R} w_{kj} - \sum_{j \in \bar{R}} w_{kj} \ .$$

3.4.3 Simulated annealing (SA)

Simulated annealing, originally presented by Kirkpatrick et al. [32], is a general purpose global optimization technique for very large scale numerical optimization problems. This method is related to the *Monte Carlo* (MC) method [33] and is based on the principles

of thermodynamics. The process simulates the common industrial process of "annealing" — obtaining a low material energy (such as less brittle glass) by slowly cooling a melted material. At the beginning, the temperature is relatively high so atoms can move freely. As the process proceeds, the temperature is carefully decreased so that atoms can move just enough to begin adopting the most stable configuration. If the cooling is done carefully enough then at the end of the process the most stable configuration is reached, i.e., the one that has the lowest energy. This basic concept can be applied to multi-variable optimization problems, in particular it is used to escape local optima. The analogy is straightforward. The current thermodynamic configuration is analogous to the current approximation to the solution of the optimization problem. The energy functional is analogous to the objective function and the final stable state of minimum energy is analogous to the optimum state.

The simulated annealing optimization starts with high temperature T. For minimization problems, any random step that reduces the energy is accepted. Since T is high, the probability to accept a random step that results in an energy increase should still be relatively high, increasing with T and decreasing with the size of the energy increase. A common used probability function that satisfies this need is $P(\Delta E) = min(1, exp(\frac{-\Delta E}{T}))$, where $\Delta E = En_{new} - En_{old}$. After sufficient number of MC steps with high temperature, the temperature T is reduced and the MC process is repeated until sufficiently small T is reached. The way T varies is known as the *cooling schedule*. It is used to control the rate of convergence and the strategy of exploring the solution space. Different cooling schedules are being used: a linear cooling schedule $T_{new} = T_{old} - \Delta T$; a proportional cooling schedule $T_{new} = C \cdot T_{old}$, where $C < 1.0$; etc. This process in large problems would usually need to apply *very gradual* cooling (decrease of temperatures), making it extremely slow and inefficient for approaching the global optimum.

In the multilevel framework, however, the role of SA is somewhat different. At each level it is assumed that the *global* partition has been inherited from the coarser levels, and thus only *local*, small-scale changes are needed. For that purpose, we have started at relatively high T, lowered it *substantially* at each subsequent sweep, until strict minimization is employed.

In particular, the SA is applied only to nodes that are located along the cut line. Each move of such a node (to the other region) is assigned with a particular probability value. Consider a node i, denote by ΔE the change in En caused by moving i from R to $\bar{R}$. The probability that this move will be accepted is:

$$Prob\_move(i) \stackrel{def}{=} \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ \exp(-\frac{\Delta E}{T \cdot S_i}) & \text{otherwise} \end{cases},$$

where S_i is the strength of the connection between i and $\bar{R}$, i.e., $S_i = \frac{\sum_{j \in \bar{R}} w_{ij}}{\sum w_{ij}}$. In other words, a move that causes a small change in En but makes a *big* change along the cut line gets a higher probability to be accepted, as it is expected that some local relaxation around the change will produce a lower partition cost. The initial temperature T is calculated a-priori for each level l by aiming at the acceptance, *accept*, of certain percentage of moves: 2% on the coarsest level which is gradually increased by equal increments up to 14% on the finest level. In particular, calculate $\frac{\Delta E}{S_i}$ for each node on the border and sort the results in ascending order. Choose the entry located at the *accept* relative location as the current value of $\frac{\Delta E}{S_i}$, denoted by $\widetilde{\Delta}$, and calculate the value of T from $0.5 = \exp\left(-\frac{\widetilde{\Delta}}{T}\right)$. Note that nodes with lower value (of $\frac{\Delta E}{S_i}$) than $\widetilde{\Delta}$ have a higher probability (than 0.5) of accepting their move (closer to 1). We used proportional cooling schedule: T is reduced to $reduce\_T\_factor$ times its previous value at each subsequent sweep with $reduce\_T\_factor = 0.7$. The localness of these changes and the rapid cooling guarantee the preservation of the large scale bisection inherited from the coarser levels. We run the SA process (heating-cooling) $num\_heating\_cooling$ times, in each iteration we first calculate an initial temperature and then decrease it iteratively num_reduce_T times. We used $num\_heating\_cooling = 20$ and $num\_reduce\_T = 5$. At the end of each cooling process we employed strict minimization relaxation.

Note that we may reach the minimal En during the SA process and not necessarily at its end and that it may even be destroyed by the next heating-cooling iteration. This difficulty is treated below. For complete details see Procedure SA.

3.4.4 Lowest common configuration (LCC)

For a large enough graph it is expected that the detailed bisection of one subgraph will often be independent of those of other subgraphs. That is, even when part of the global minimal bisection has been reached, it might be later destroyed (by using further heating) before other parts attain their minimal bisection values. It might, of course, take a very long time before all parts simultaneously settles at the optimal cut. This slowness can, however, be overcome by adding some "memory" to the annealing process. More precisely, instead of just storing (one or several) *best-so-far* (BSF) visited bisections, a possibly better bisection than all of these can be obtained using the so called *Lowest Common Configuration* (LCC) procedure introduced by Brandt et al. in [10] and by Ron in [36], and applied by Safro et al. [38] in the context of the linear arrangement problem. In our context the LCC procedure applied on two given bisections basically selects partial sub-bisections out of the two given ones and forms a third bisection with En not higher than its two constituents. The BSF, of

a certain level, is initialized by the bisection obtained at the end of the strict minimization. Then the BSF is improved by the LCC procedure which updates parts of it taken from the new bisections reached either during the SA process, if the value of En happens to be lower than that of the BSF, or at the end of each heating-cooling procedure. This, of course, justifies the repeated re-heating process involved in the annealing with no danger of possible damage to already achieved parts of the global minimal partition. When the entire disaggregation process is completed, the accumulated BSF serves as the bisection of the current fine level.

The LCC procedure works as follows. Suppose that after a step of the heating-cooling procedure we have reached a new bisection. The LCC marks the set of nodes which lie in different regions with respect to the current BSF and the new bisection. Then, a set of connected clusters of the marked nodes is created (two marked nodes which share an edge are in the same cluster). For every cluster k, all nodes $i \in k$ are temporarily put in a region different from their current BSF assignment. The value of En of this "new" assignment is then calculated. In case a lower value of En is reached, BSF is modified to assign all nodes $i \in k$ to their new region. Otherwise, BSF remains unchanged. To demonstrate the benefits of using LCC we ran the algorithm both with and without it, and compared the results (see Section 4.3). The LCC procedure as written below runs in a linear time with respect to the number of edges and the number of vertices, i.e., $O(|V|+|E|)$. For complete details see Procedure LCC.

Procedure LCC**(BSF, bisection of** G**)**
$current\_En \leftarrow En$ of the BSF configuration
Mark all nodes with different assignments in BSF and in the bisection of G
While there are still marked nodes
    Pick a marked node i
    $current\_cluster \leftarrow$ the cluster of all marked nodes connected to i
    Unmark all the nodes appearing in $current\_cluster$
    $new\_En \leftarrow En$ of BSF when we flip the assignment of all nodes $i \in current\_cluster$
    If $(new\_En \leq current\_En)$
        Accept the changes into BSF and update the value of $current\_En$
End.

An overview of the entire disaggregation process is given in Procedure Disaggregation.

Procedure SA(bisection of G,$num\_heating\_cooling$**,**num_reduce_T**,**$reduce\_T\_factor$**)**
$BSF \leftarrow$bisection of G
Repeat $num\_heating\_cooling$ times
    $current\_T \leftarrow$ calculate $initial\_T$ (for details see Section 3.4.3)
    Repeat num_reduce_T times
      For all nodes $i \in R$ that have a neighbor in $\bar{R}$
            $En \leftarrow$ current En
            $new\_En \leftarrow En$ when i is moved to $\bar{R}$
            $\Delta En \leftarrow new\_En - En$
            $S_i \leftarrow \sum_{j \in \bar{R}} w_{ij}$
            Move i to $\bar{R}$ with probability $min(1, e^{-\frac{\Delta E}{S_i \cdot T}})$
      $current\_T \leftarrow reduce\_T\_factor \cdot current\_T$
    Apply $num\_minimization$ sweeps of strict minimization on G_f
    $BSF \leftarrow LCC(BSF,$ bisection of $G)$
Return BSF
End.

Procedure Disaggregation(coarse graph G_c**, fine graph** G_f**)**
Parameters: $num\_minimization$,$num\_heating\_cooling$,num_reduce_T,$reduce\_T\_factor$
Initialize G_f (see Section 3.4.1 for details)
Apply $num\_minimization$ sweeps of strict minimization on the F-nodes of G_f
Apply $num\_minimization$ sweeps of strict minimization on G_f
SA(bisection of G_f,$num\_minimization$,$num\_heating\_cooling$,num_reduce_T,$reduce\_T\_factor$)
End.

3.4.5 Bisections elimination process based on efficiency considerations

In order to explore different possible bisections, we start by choosing $num\_coarse\_solutions$ different bisections from the coarsest level (see Section 3.3). During the SA process (especially at coarser levels) the cut might change and become similar to one of the bisections we already have. Therefore we might want to keep the bisections before the SA as well. In order to limit the running time of the algorithm, the number of different bisections that we propagate back to the finest level must be restricted. We measure the amount of work at each level by the number of edges. Note, that this number might increase at the beginning of the coarsening process, i.e., at the first two or three levels. However, due to the decrease in the number of nodes and the dilution used as the coarsening proceeds (see Section 3.2.3), the number of edges eventually decreases. The maximum number of bisections that will be propagated all the way back to the finest level is restricted to $num\_fine\_solutions$. This is also true for every level i that satisfies $|E_i| \geq \frac{1}{2} \cdot |E_0|$. As the number of edges in the graph decreases ($|E_i| < \frac{1}{2} \cdot |E_0|$), the allowed amount of work is reduced by a factor of τ. That is, the maximum number of different bisections that will be propagated to the i'th level, $num\_i\_solutions$ is:

$$num\_i\_solutions = \begin{cases} num\_fine\_solutions & \forall i \leq m \\ min[2 \cdot num\_coarse\_solutions, \frac{|E_0| \cdot num\_fine\_solutions}{|E_i|} \cdot \tau^{i-m}] & \forall i > m \end{cases},$$

where $m = arg\ max_j(|E_j| \geq \frac{1}{2}|E_0|)$. We chose to limit the number of bisections as follows: $num\_corase\_solutions = 40$, $num\_fine\_solutions = 5$, while $\tau = \frac{2}{3}$. Bisections are considered to be different if Equation 6 is satisfied using variable ω so as to maintain $num\_i\_solutions$ bisections. Finally, LCC was performed on the $num\_fine\_solutions$ obtained for the original graph, to produce their best combination. This is the final result of the algorithm.

4 Numerical results

Section 4.1 describes the used test suite and introduces a new set of contrived test case graphs. Section 4.2 provides information about available software tools for graph partitioning. Last, in Section 4.3 we bring a summary of our results [48] compared with other works.

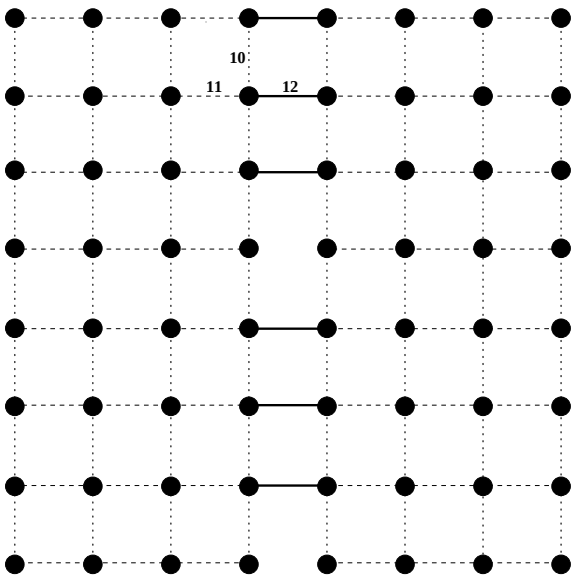

Figure 2: A mesh graph: The bold horizontal edges are of strength 12, the dashed horizontal ones 11; the vertical dotted edges 10. Any straight horizontal bisectioning costs 80, any vertical non-centric costs 88, while the central vertical cut, the global minimum, costs only 72.

4.1 Test case graphs

Experimental tests have been performed on a wide range of graphs of different sizes and topologies taken from [45]. The characteristics of these graphs are described in table 1. It includes 2D and 3D examples of *nodal graphs*, where the finite element mesh nodes are partitioned and *dual graphs*, where the mesh elements are partitioned. In addition there is a 3D semi-structured graph, cti, and non mesh based graphs which arise from various scientific computing applications such as add32, vibrobox, bcsstk30 etc. None of these graphs have either vertex or edge (non-uniform) weights.

In addition we have constructed some more graphs for the following reason. The main difficulty of finding the global minimum for a discrete variables problem is caused by its complicated landscape structure which involves a multitude of local minima usually nested within each other. These attraction basins result from the local structure of the problem, e.g., local, strong or dense connections being present in subgraphs of the given graph. Any clustering that is "greedy", and thus would usually prefer the matching of the strongest local connections, would tend to assign such a substructure fully to one of the partitions. If, however, the global minimum is only obtained by partitioning such subgraphs, these algorithms would most likely be trapped in a local minimum. The algorithm suggested in this report attaches such local connections only partially and avoids making hardened local

decisions, such as edge contractions, before accumulating the relevant global information. Observe for example the mesh graph presented in Figure 2, where the bold horizontal edges are of strength 12, the dashed horizontal ones are 11 and the vertical dotted edges are 10. Clearly, all straight horizontal bisections cost $8 \times 10 = 80$, while all vertical non-centric cost $8 \times 11 = 88$. However, the central vertical cut which is the global minimum costs only $6 \times 12 = 72$. We have created similar mesh graphs of sizes 200×200, 400×400 and 800×800, with increased values of the "horizontal/vertical ratio", ψ, which is defined by

$$\psi = \frac{\text{local minimum: horizontal cutsize}}{\text{global minimum: centric vertical minimum cutsize}} .$$

The value of ψ was increased by deleting a larger number of vertical edges from the central column of the mesh.

4.2 Available software tools for graph partitioning

During the last few years many public domain software tools have been developed for graph partitioning. In this section we list some available software packages.

- **CHACO**, [23]. Developed by Hendrickson and Leland at Sandia National Labs. It contains implementation of the inertial, spectral, Kernigham-Lin and multilevel-KL algorithms.
- **TOP/DOMDEC**, [16]. Developed by Simon Farhat and Lanteri. It includes implementation of the greedy, recursive graph bisection, inertial and recursive spectral bisection algorithms.
- **JOSTLE**. Developed by Walshaw et al. at Univ. of Greenwich. It includes implementation of the greedy [15], recursive coordinate bisection [41], multilevel Kernighan-Lin [47], iterated multilevel Kernighan-Lin [46] and combined evolutionary/multilevel [42] algorithms.
- **METIS**. Based on the work of Karypis and Kumar [26, 29, 28] at the Dept. of Computer Science, Univ. of Minnesota. They have later extended this package to a hypergraph partitioning software package named *hypergraph-METIS* (hMETIS) [27], by implementing the algorithms described in [30, 25].

The software-packages / algorithms that were used to get the minimal cut on the test suite described in Table 1 are abbreviated in table 2.

Graph Name	|V|	|E|	min/avg/max degree	type
add20	2395	7462	1/6.23/123	20-bit adder
data	2851	15093	3/10.58/17	3D nodal graph
3elt	4720	13722	3/5.81/9	2D nodal graph
uk	4824	6837	1/2.83/3	2D dual graph
add32	4960	9462	1/3.81/31	32-bit adder
bcsstk33	8738	291583	19/66.73/140	3D stiffness matrix
whitaker3	9800	28989	3/5.91/8	2D nodal graph
crack	10240	30380	3/5.93/9	2D nodal graph
wing_nodal	10937	75488	5/13.80/28	3D nodal graph
fe_4elt2	11143	32818	3/5.89/12	Not indicated
vibrobox	12328	165250	8/26.8/120	vibroacustic matrix
bcsstk29	13992	302748	4/43.27/70	3D stiffness matrix
4elt	15606	45878	3/5.87/10	2D nodal graph
fe_sphere	16386	49152	4/5.99927/6	Not indicated
cti	16840	48232	3/5.72/6	3D semi-structured graph
memplus	17758	54196	1/6.10/573	digital memory circit
cs4	22499	43858	2/3.90/4	3D dual graph
bcsstk30	28924	1007284	3/69.65/218	3D stiffness matrix
bcsstk31	35588	527914	1/32.19/188	3D stiffness matrix
fe_pwt	36519	144794	0/7.92/15	Not indicated
bcsstk32	44609	985046	1/44.16/215	3D stiffness matrix
t60k	60005	89440	2/2.98/3	2D dual graph
wing	62032	121544	2/3.91/4	3D dual graph
brack2	62631	366559	3/11.70/32	3D nodal graph
fina512	74752	261120	2/6.98/54	Linear programming
fe_tooth	78136	452591	3/11.58/39	Not indicated
200×200	79950	40000	2/3.97/4	2D nodal graph
400×400	319094	160000	2/3.98/4	2D nodal graph
800×800	5116365	2560000	2/3.99/4	2D nodal graph

Table 1: Description of the test suite.

4.3 Results summary

Table 3 summarizes the results obtained by running the most popular up-to-date partitioning algorithm for hypergraphs, the hMETIS [27], on the test suite described in Table 1.

Abbreviation	Software	References
Greedy	Farhat's greedy algorithm (within JOSTLE)	[15]
RCB	Recursive Coordinate Bisection (within JOSTLE)	[41]
MRSB	Bernard and Simon's Multilevel Recursive Spectral Bisection	[3]
Ch2.0	CHACO - version 2.0	[23]
M4.0	METIS - version 4.0	[29, 28]
GTS	Stephane Popinet's multilevel implementation available as part of the GNU Triangulated Surface Library	——
J2.2	Multilevel Kernighan-Lin (k-way) (within JOSTLE)	[47]
iJ	iterated multilevel Kernighan-Lin (k-way) (within JOSTLE)	[46]
JE	Combined evolutionary / multilevel scheme (within JOSTLE)	[42]
GrPart	Alexander Kozhushkin's implementation of iterative multilevel Kernighan-Lin	——
MLSATS	MultiLevel refinated Mixed Simulated Annealing and Tabu Search	[2]

Table 2: Abbreviations of the software packages/algorithms that were used to get the minimal cut. Taken from [45].

The hMETIS program may be executed with different arguments. In order to achieve the best possible results, we used five different coarsening schemes combined with two different refinement schemes. For each of these ten combinations we have executed the program one hundred times. In Table 3 we present the *best* results (out of the ten possibilities) obtained by hMETIS along with the results obtained by our algorithm averaged over ten runs using different sequences of random numbers, i.e., ten different disaggregations applied to the same aggregation. We have not included any execution time results since our algorithm have not yet been optimized in this respect.

Looking at Table 3, one can observe that the minimum cost obtained using our algorithm is comparable to that obtained using hMETIS for all graphs except for Add20 and Memplus. Excluding these two graphs, in 18 out of 26 graphs the minimal cut found was the same for both algorithms. In 2 test graphs the minimal cut found by hMETIS was higher by an average of 2.24%. In 6 test graphs the minimal cut found by our algorithm was higher by an average of 0.35% than hMETIS. In 24 graphs the average result obtained by our algorithm was lower than hMETIS. Furthermore, in general the standard deviation obtained by our algorithm was significantly lower, i.e., our algorithm is less sensitive to the choice of sequences of random numbers.

For the Add20 and the Memplus test graphs the minimal cut obtained by our algorithm

Graph	hMETIS			Our algorithm		
	average	SDV	min. cut	average	SDV	min. cut
add20	**678.14**	57.73	**577**	704.2	0.63	704
data	198.8	9.26	**186**	**190.9**	**3.81**	187
3elt	88.75	1.43	**87**	**87.3**	**0.67**	**87**
uk	20.25	1.50	**18**	**19**	**0.94**	**18**
add32	10.3	1.169	**10**	**10**	**0**	**10**
bcsstk33	10100.64	185.56	**10064**	**10064**	**0**	**10064**
whitaker3	128.55	1.81	**126**	**127.2**	**0.91**	**126**
crack	188.64	5.92	**182**	**183.5**	**2.12**	**182**
wing_nodal	1756.11	59.082	**1688**	**1691**	**2.49**	1690
fe_4elt2	**130**	**0**	**130**	**130**	**0**	**130**
vibrobox	11303.08	484.56	**10310**	**10310**	**0**	**10310**
bcsstk29	2853.68	18.271	**2818**	**2819**	**0**	2819
4elt	142.8	8.277	**138**	**138**	**0**	**138**
fe_sphere	**384**	**0**	**384**	384.2	0.63	**384**
cti	330.62	17.49	**318**	**318**	**0**	**318**
memplus	**6044.25**	408.25	**5413**	6474.7	116.31	6300
cs4	399.2	16.192	375	**366.7**	**1.15**	**365**
bcsstk30	6354.94	168.197	**6251**	**6297.3**	**30.44**	**6251**
bcsstk31	2845.77	257.24	**2676**	**2695.2**	**11.64**	2677
fe_pwt	358.18	6.26	**340**	**340**	**0**	**340**
bcsstk32	5487.01	474.766	**4667**	**4667**	**0**	**4667**
t60k	81.51	7.49	**73**	**77.1**	**2.02**	**73**
wing	863.91	24.04	813	**799.9**	**1.59**	**799**
brack2	**705.74**	8.74	**700**	715.6	6.78	707
fina512	**162**	**0**	**162**	**162**	**0**	**162**
fe_tooth	4105.95	224.76	**3813**	**3850.8**	**18.81**	3827
200× 200	1998	20	**1800**	**1800**	**0**	**1800**
400×400	3999.56	21.97	**3528**	**3528**	**0**	**3528**
800×800	7990.2	98	**7020**	**7020**	**0**	**7020**

Table 3: Comparative table of results: Ours versus hMETIS with up to 1% imbalance. The obtained minimum average, standard deviation and cut are bolded.

was higher by 22% and 16%, and the average was higher by 3.8% and 7.1%. Add20 and Memplus belong to a class of graphs for which the coarsening scheme, as implemented now, might indeed fail. This class contains graphs in which the average degree is much smaller than the maximum degree, i.e, only few nodes are connected to many nodes. The highest degree nodes are expected to aggregate around them a large volume. Thus, during the coarsening phase they will be chosen as seeds, and will continue to be such as the process proceeds. However, the optimal cut does not necessarily divide those heavy nodes into two groups. Thus we might not be able to get the optimal cut. This problem arises from the following asymmetry appearing at coarsening: A heavy node always tends to be a seed, while a light node tends to be attached to a heavy seed to which it is strongly connected, while this heavy node might only be *weakly* connected to that light node.

For example, let $G = (V, E)$ be a graph with the following properties:

- Define A and B such that $V = A \cup B$, $A \cap B = \emptyset$,
- $|A| << |B|$,
- $\forall\ i \in B,\ j \in A\ :\ \ deg(i) << deg(j)$,
- the vertices in A are each strongly connected to each other (i.e., form a clique-like structure).

During the coarsening phase, the heavy nodes from A will be chosen as seeds. As a result, the current version of our algorithm will tend to draw the partition within this clique-like structure of aggregates. However, this is not necessarily the optimal bisection. The minimal cut may well pass through the light nodes while the set of heavy nodes is (mostly) placed at one side. See Figure 3.

Best bisections for the test suite's graphs are located and updated at Walshaw's web site [45]. These bisections (indicating the minimal cut size) classified by four levels of imbalances: $0\%, 0.5\%, 1.5\%$ and 2.5%. Tables 4, 5 and 6 compare the costs obtained by our algorithm with the minimum reported costs for $0.5\%, 1.5\%$ and 2.5%, respectively.

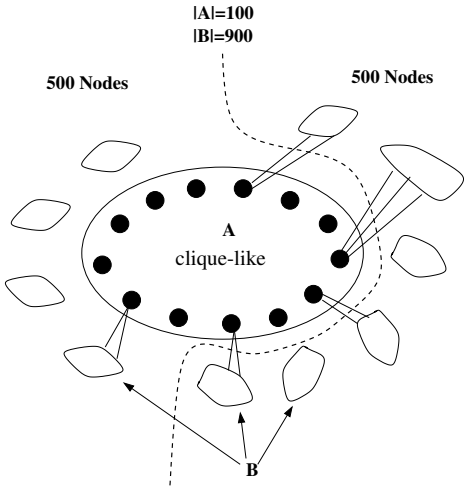

Figure 3: Example of failure of the coarsening phase. The dashed line bisect the graph into two equal sized subsets. Any cut that passes through A, which has a clique-like structure, has a much higher cost.

	Graph archive				Graph archive		
graph	Software Package/ algorithm (See Table 2)	Minimum cost	our cost	graph	Software Package/ algorithm (See Table 2)	Minimum cost	our cost
add20	GrPart	**618**	705	fe_sphere	JE	**386**	**386**
data	GrPart	196	**188**	cti	JE	**318**	**318**
3elt	GrPart	**89**	90	memplus	JE	**5492**	6471
uk	GrPart	21	**19**	cs4	JE	**367**	369
add32	J2.2	**10**	**10**	bcsstk30	GrPart	**6335**	6347
bcsstk33	GrPart	10109	**10097**	bcsstk31	GrPart	**2701**	2724
whitaker3	JE	**126**	**126**	fe_pwt	GrPart	**340**	**340**
crack	JE	184	**182**	bcsstk32	JE	**4667**	**4667**
wing_nodal	GrPart	1703	**1697**	t60k	iJ	78	**77**
fe_4elt2	MRSB	**130**	**130**	wing	JE	798	**796**
vibrobox	JE	**10310**	**10310**	brack2	GrPart	**708**	**708**
bcsstk29	GrPart	**2818**	2826	fina512	Ch2.0	**162**	**162**
4elt	GrPart	**138**	**138**	fe_tooth	GrPart	3982	**3827**

Table 4: A comparison between the "best known" results reported in [45] and our results with up to 0.5% imbalance.

graph	Graph archive: Software Package/ algorithm (See Table 2)	Graph archive: Minimum cost	our cost	graph	Graph archive: Software Package/ algorithm (See Table 2)	Graph archive: Minimum cost	our cost
add20	GrPart	**618**	702	fe_sphere	JE	**384**	**384**
data	GrPart	196	**185**	cti	JE	**318**	**318**
3elt	JE	**87**	**87**	memplus	JE	**5407**	6404
uk	JE	**18**	**18**	cs4	JE	363	**362**
add32	J2.2	**10**	**10**	bcsstk30	JE	**6251**	**6251**
bcsstk33	GrPart	10096	**10064**	bcsstk31	MLSATS	**2676**	**2676**
whitaker3	JE	**126**	**126**	fe_pwt	GrPart	**340**	**340**
crack	JE	184	**182**	bcsstk32	JE	**4667**	**4667**
wing_nodal	JE	1686	**1681**	t60k	JE	72	**71**
fe_4elt2	MRSB	**130**	**130**	wing	JE	**778**	780
vibrobox	JE	**10310**	**10310**	brack2	JE	**684**	686
bcsstk29	GrPart	**2818**	**2818**	fina512	Ch2.0	**162**	**162**
4elt	JE	**137**	138	fe_tooth	GrPart	3982	**3829**

Table 5: A comparison between the "best known" results reported in [45] and our results with up to 1.5% imbalance.

graph	Graph archive: Software Package/ algorithm (See Table 2)	Graph archive: Minimum cost	our cost	graph	Graph archive: Software Package/ algorithm (See Table 2)	Graph archive: Minimum cost	our cost
add20	GrPart	**618**	702	fe_sphere	JE	**384**	**384**
data	GrPart	196	**182**	cti	JE	**318**	**318**
3elt	JE	**87**	**87**	memplus	iJ	**5353**	6252
uk	JE	**18**	**18**	cs4	JE	363	**356**
add32	J2.2	**10**	**10**	bcsstk30	JE	**6251**	**6251**
bcsstk33	iJ	**9914**	**9914**	bcsstk31	MLSTAS	**2676**	**2676**
whitaker3	JE	**126**	**126**	fe_pwt	GrPart	**340**	**340**
crack	JE	184	**182**	bcsstk32	JE	**4667**	**4667**

	Graph archive				Graph archive		
	Graph archive				Graph archive		
graph	Software Package/ algorithm (See Table 2)	Minimum cost	our cost	graph	Software Package/ algorithm (See Table 2)	Minimum cost	our cost
wing_nodal	MLSATS	**1670**	1672	t60k	JE	72	**67**
fe_4elt2	MRSB	**130**	**130**	wing	JE	778	**774**
vibrobox	JE	**10310**	**10310**	brack2	MLSATS	668	**661**
bcsstk29	GrPart	**2818**	**2818**	fina512	Ch2.0	**162**	**162**
4elt	JE	**137**	**137**	fe_tooth	GrPart	3982	**3805**

Table 6: A comparison between the "best known" results reported in [45] and our results with up to 2.5% imbalance.

From Tables 4, 5 and 6 we conclude that our results are comparable to (and in some cases even better than) those obtained by the best other software packages/algorithms. In particular:

- For up to 0.5% imbalance (Table 4), in 7 out of 26 test graphs (including Add20 and Memplus) our minimal cut was higher than the minimal "known" cut by an average of 4.98%. If we exclude Add20 and Memplus this difference decreases to 0.6%. In 8 out of 26 test graphs we found a cut that was on average 2.56% lower than the minimum previously known.

- For up to 1.5% imbalance (Table 5), in 5 out of 26 test graphs (including Add20 and Memplus) our minimal cut was higher than the minimal "known" cut by an average of 6.7%. If we exclude Add20 and Memplus this difference decreases to 0.43%. In 7 out of 26 test graphs we found a cut that was on average 1.83% lower than the minimum previously known.

- For up to 2.5% imbalance (Table 6), in 3 out of 26 test graphs (including Add20 and Memplus) our minimal cut was higher than the minimal "known" cut by an average of 10.17%. If we exclude Add20 and Memplus this difference decreases to 0.12%. In 6 out of 26 test graphs we found a cut that was on average 3.30% lower than the minimum previously known.

Table 7 summarizes the results obtained by running the hMETIS [27] on the different meshes we have constructed. In order to get the best possible hMETIS results we combined

five different coarsening schemes with two different refinement schemes. In addition, we allowed imbalance of one percent and two percents between the two parts. For every such combination we ran the program one hundred times. Hence, the total number of hMETIS trials was *two thousands*. As shown in Table 7 hMETIS's success rate is very small, i.e., less than 0.3%. On the other hand, our algorithm always finds the minimal cut, i.e., the average result equals to the minimal cut and the standard deviation is zero (see Table 3).

Mesh size	$\|E\|$	$\|V\|$	Min. cut cost	ψ	hMETIS		
					Number of successes	average	SDV
200×200	79950	40000	1800	1.11	4	1999.82	13.298
400×400	319094	160000	3528	1.13	3	3999.56	21.971
800×800	1278185	640000	7020	1.139	7	7998.03	72.120

Table 7: Results obtained by running hMETIS on different meshes

To demonstrate the benefits of using LCC we ran the algorithm both with and without it, and compared the results. Table 8 summarizes the average results obtained by running ten tests with different choices of sequences of random numbers. From Table 8, we conclude that the results obtained by using SA combined with LCC are better by an average of 7.27% than the results obtained by using SA only. The percentage by which the average result is improved by employing the LCC is given in the bottom line of Table 8 separately for each imbalance.

graph	0.5% imbalance		1% imbalance		1.5% imbalance		2.5% imbalance	
	with LCC	without LCC	with LCC	without LCC	with LCC	without LCC	with LCC	without LCC
add20	708.9	725	704.2	723.4	705.7	719.7	697.3	723.7
data	194.9	195	190.9	193.8	189.5	190.5	184.5	193.8
3elt	90	92.9	87.3	88.7	87	90.6	87	90.4
uk	19	22.3	19	21.4	18	21.8	18	21.2
add32	10	10	10	10	10	12.8	10	11
bcsstk33	10102.2	10122.9	10064	10073.4	10064.3	10066	9914	9974
whitaker3	127.7	138.3	127.2	128.9	126.3	129.2	126	127.9
crack	184.8	192.1	183.5	191.8	183.5	192.3	182.3	190.8
wing_nodal	1700	1717.4	1691	1708.9	1686	1702.2	1673.7	1680.4
fe_4elt2	130	130	130	130	130	130	130	130

graph	0.5% imbalance		1% imbalance		1.5% imbalance		2.5% imbalance	
	with LCC	without LCC	with LCC	without LCC	with LCC	without LCC	with LCC	without LCC
vibrobox	10310	10842.7	10310	10693.5	10310	10720.4	10310	10800.9
bcsstk29	2826	2890.6	2819	3020.9	2818.6	3030.3	2822.5	2927.3
4elt	138.9	138.7	138	138.8	138	138.2	137.8	138.7
fe_sphere	386	390.8	384.2	390.8	385	390.4	385	390.2
cti	318	367.6	318	346.8	325.5	350	324.1	325.3
memplus	6493.8	6562.9	6474.7	6567.1	6621	6712.3	6452.4	6702.6
cs4	370.4	390.4	366.7	388.3	364.3	385.5	357.4	381.9
bcsstk30	6347.6	6398.3	6297.3	6315.6	6258.1	6304.9	6284.1	6307.7
bcsstk31	2741.3	2845.3	2695.2	2864.8	2683.1	2862.1	2676.1	2874.5
bcsstk32	4667	4754.7	4667	4703.2	4667	4711.4	4667	4712.4
t60k	80.8	127.1	77.1	112.5	71.6	116	68.6	110.2
wing	801	842.4	799.9	850.5	791.4	838.9	780.5	846
brack2	719.1	722.7	715.6	728.9	716	730.1	670.8	678
fe_tooth	3878.7	4141.6	3850.8	4106.2	3869	4067.6	3863.4	3989.7
200×200	1800	2068.3	1800	2076.8	1800	2064.7	1800	2071.5
400×400	3528	4578.1	3528	4484	3528	4452.1	3528	4270.8
LCC Improvement	7.22%		6.35%		8.30%		7.21%	

Table 8: Results obtained by running our algorithm with and without LCC.

5 Conclusions and future work

We have presented a new multilevel algorithm for the bisectioning problem for sparse graphs. The algorithm is based on the general principle that during the coarsening each vertex may be associated to more than just one aggregate according to some "likelihood" measure. The uncoarsening initialization, which produces the first bisectioning of the fine graph nodes, strongly relies on energy considerations (unlike usual interpolation in classical AMG). This initial bisectioning is further improved by local strict minimization relaxation and by employing stochasticity, i.e., simulated annealing, which is enhanced by the LCC procedure. The algorithm suggests a new energy function that reflects the tradeoff between the bisection problem's two objectives: the cut minimization and the balance constraint. In order to get a lower cut size it allows increased imbalance, which is gradually reduced as the disaggregation proceeds. The running time of the algorithm is linear, thus it can be applied to very large graphs.

Except for a special class of graphs in which there exists a subgraph which is strongly connected within itself, the minimum cost obtained using our algorithm is practically equal to (and in some cases even better than) the best known results. Furthermore, the average and standard deviation are lower than the ones achieved by hMETIS (the most popular up-to-date partitioning algorithm), i.e., our algorithm is less sensitive to the choice of sequences of random numbers.

To show the advantage of using weighted aggregation rather than strict aggregation we have constructed a class of graphs in which the global optimum contradicts the local information. We claimed that any clustering that is "greedy" would usually prefer the matching of the strongest local connections and thus would fail to find the optimal bisection. When using weighted aggregation such local connections will only be partially attached and while taking into consideration more global information the process would finally reach the optimal bisection. This claim was supported by experimental tests on hMETIS and our algorithm.

Besides the above positive indications of this preliminary version of our bisectioning algorithms there still remain a lot of work that could be done:

- Implement the algorithm in a way that minimizes the running time so as to get comparable running time.

- We mentioned above that the algorithm does not work for a special class of graphs in which there exists a subgraph which is strongly connected within itself. This problem arises from an asymmetric coarsening when a heavy vertex always tends to be a seed and a light vertex tends to be attached to a heavy seed to which it is strongly connected, while the heavy vertex is only *weakly* connected to that light vertex. A possible way to correct this feature is to develop a strategy of combining several light vertices into a special kind of aggregate that can be compared with "naturally" heavy vertices.

- Currently, the algorithm uses a simple strict node by node minimization along the cut line: Given a node $i \in R$ which is located along the cut (i.e., there exists some edge ij such that $j \in \bar{R}$), the new energy associated with the assignment of i to region $\bar{R}$ is then calculated, and if the result is lower or equal to the old energy, the change is accepted. This could be extended by trying to move each such node and *revise* around it, i.e., find out what happens if we moved along with it some of its neighbors either in this level or possibly on finer levels and only then decide whether to accept the entire move (similar to the strategy in [10]). It would be interesting to

check the effect of this generalization on the algorithm. It might explore new possible bisections.

- Generalize the algorithm to a k-way partitioning.
- Generalize the algorithm to hypergraphs partitioning.

6 Acknowledgements

The research has been supported by grants No. 295/01 from the Israel Science Foundation, US Air Force Office Of Scientific Research contract F33615-03-D-5408 and by a Grant from the German-Israeli Foundation for Scientific Research and Development (G.I.F.), Research Grant Agreement No. I-718-135.6/2001.

APPENDIX

Overview of the parameters used by the algorithm

In this section we summarize the setting of all parameters used by our algorithm. We use the convention that if parameters are set to a fixed value, then that value appears in parentheses following the parameter's name.

- **imbalance_factor$_i$** - Used by Equation 3 to calculate $En(G_i)$. Its value indicates the imbalance allowed at the i'th level, as a percentage. At the finest level, it equals to the imbalance factor given by the user. At any other level $i > 0$, it equals to the relative volume of the heaviest node, i.e., $imbalance\_factor_i = \max\left(imbalance\_factor, \frac{v_{j(i)}}{|V_0|}\right)$, where $v_{j(i)}$ is the volume of the heaviest node at level i.
- **initial_punish_cost**(0.10), **border_punish_cost**(0.02) - These parameters are used to determine the value of ρ appearing in Equation 3. Namely, ρ is determined by the requirement that if the deviation from the $imbalance\_factor_i$ (see above) equals half of the volume of one of the heaviest nodes, then the energy will be larger than the cutsize by either $initial\_punish\_cost$ or $border\_punish\_cost$. More precisely, at each level, before a bisection is initialized, ρ is determined by the following equation

$$1 + initial\_punish\_cost = exp\left(\rho \cdot \frac{0.5 \cdot v_{max}}{|V_0|} \cdot 100\right),$$

where v_{max} is the volume of the node with maximum volume. After an initial bisection, ρ is determined by

$$1 + border\_punish\_cost = exp\left(\rho \cdot \frac{0.5 \cdot v_{max_border}}{|V_0|} \cdot 100\right),$$

where v_{max_border} is the volume of the node with maximum volume along the cut.

- $\eta(2)$ - Used in Procedure Fix_C_Points. A fine node i with a future volume larger than this factor times the average future volume is automatically added to C.
- $\nu(8)$ - Used in Procedure Fix_C_Points. It indicates the number of compatible GS relaxation sweeps.
- **num_groups**(3) - Used in Procedure Fix_C_Points. It determines to how many groups to divide the outcome of the GS relaxation from which the seeds are then chosen.
- **Q**(0.4) - Used in Procedure Fix_C_Points. Fine nodes that have a relative total connection to C lower than this threshold, are add to C with higher priority.
- $\mathbf{r_0}(6)$ - Used when calculating the aggregation weights. It is the interpolation order used for the finest level.
- **R** - Used when calculating the aggregation weights. At level L, it equals $\log(min(1, |E_0|/|E_L|))$.
- **r** - Used when calculating the aggregation weights. It is the interpolation order parameter, which equals r_0 at the finest level. At any other level L, it has the value $r_0 + R$.
- $\epsilon(\mathbf{0.001})$ - Used at the end of each coarsening phase: Edge filtering threshold, aimed at reducing the complexity of the algorithm.
- ω - Used in the bisections elimination process and when solving the coarsest level. Two bisections are considered to be different if the relative total volume of nodes in which they differ is greater than this parameter (Equation 6). At the coarsest level, it equals 0.08. At any other level i during the bisections elimination process it has a variable value so as to maintain *num_i_solutions* (see below).
- **num_coarse_solutions**(40) - Used in the bisections elimination process and when solving the coarsest level: The number of different bisections chosen from the coarsest level. (Although we fixed it to 40, a perhaps better way would be to choose it proportional to the original graph size.)
- **certainty** - Used in the process of initialization by layers. Nodes with relative connection to a region R higher than this parameter are assigned first. It is initialized by 0.95 and reduced to 0.90.

- **num_minimization**(10) - Used in the disaggregatoin and in the simulated annealing processes. It indicates the allowed maximal number of strict minimization sweeps; i.e., if En has not been reduced in the last sweep, minimization is aborted.
- **num_heating_cooling**(20) - Used in the simulated annealing process: The number of times we execute the simulated annealing process.
- **accept** - Used in the simulated annealing process, it indicates the expected percentage of non decreasing energy moves that will be accepted. It is initialized to 2% at the coarsest level, then gradually increased by equal increments until it reaches 14% at the finest level.
- **num_reduce_T**(5) - Used in the simulated annealing process, it indicates the number of times the temperature is being decreased for each heating-cooling iteration.
- **reduce_T_factor**(0.7) - Used in the simulated annealing process: The factor by which the temperature is reduced at each step of the simulated annealing process.
- $\tau(\frac{2}{3})$ - Used in the bisections elimination process: The factor by which the estimated amount of work is reduced with respect to the former (finer) level.
- **num_fine_solutions**(5) - Used in the bisections elimination process: The maximum number of bisections that will be propagated to the finest level.
- **num_i_solutions** - Used in the bisections elimination process: The maximum number of bisections that will be propagated to the i'th level:

$$num\_i\_solutions = \begin{cases} num\_fine\_solutions & \forall i \leq m \\ min[2 \cdot num\_coarse\_solutions, \frac{|E_0| \cdot num\_fine\_solutions}{|E_i|} \cdot \tau^{i-m}] & \forall i > m \end{cases},$$

where $m = arg\ max_j(|E_j| \geq \frac{1}{2}|E_0|)$.

References

[1] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *Integration: The VLSI J.*, 19:1–18, 1995.

[2] R. Banos, C. Gil, J. Ortega, and F. G. Montoya. Multilevel heuristic algorithm for graph partitioning. In *Applications of Evolutionary Computing, EvoWorkshops2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM*, volume 2611 of *LNCS*, pages 143–153, University of Essex, England, UK, 14-16 April 2003. Springer-Verlag.

[3] S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection. In *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 711–718, Philadelphia, 1993. SIAM.

[4] A. Brandt. Algebraic multigrid theory : The symmetric case. *Appl. Math. Comput.*, 19:23–56, 1986.

[5] A. Brandt. General highly accurate algebraic coarsening. *Elect. Trans. Numer. Anal.*, 10:1–20, May 1999.

[6] A. Brandt. Multiscale scientific computation: Review 2001. In T. Barth, R. Haimes, and T. Chan, editors, *Proc. of the Yosemite Educational Symposium on Multiscale and Multiresolution methods*, 2000.

[7] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations. Technical report, Institute for Computational Studies, Fort Collins, CO, POB 1852, 1982.

[8] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for sparse matrix equations. In D.J. Evans, editor, *Sparsity and its Applications*, pages 257–284. Cambridge University Press, 1984.

[9] A. Brandt and D. Ron. Multigrid solvers and multilevel optimization stratagies. In J. Cong and J.R. Shinnerl, editors, *Multilevel Optimization in VLSICAD*. Kluwer, 2002.

[10] A. Brandt, D. Ron, and D. J. Amit. Multi-level approaches to discrete-state and stochastic problems. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods II*, pages 66–99, Berlin, Heidelberg, New York, Tokyo, 1986. Springer-Verlag.

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

[12] T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Transactions On Computers*, 45(7):841–855, 1996.

[13] F. Cao, J. R. Gilbert, and S. Teng. Partitioning meshes with lines and planes. Technical Report CSL-96-01, Parc Xerox, January 1996.

[14] A. E. Dunlop and B. W. Kernighan. A procedure for placement of standard-cell VLSI circuits. *IEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, CAD-4(1):92–98, January 1985.

[15] C. Farhat. A simple and efficient automatic FEM domain decomposer. *Computers and Structures*, 28(5):579–602, 1988.

[16] C. Farhat, S. Lanteri, and H. D. Simon. TOP/DOMDEC a software tool for mesh partitioning and parallel processing. *Comput. Syst. Eng.*, 6:13–26, 1995.

[17] Feige and Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SICOMP: SIAM Journal on Computing*, 31, 2002.

[18] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *ACM IEEE Nineteenth Design Automation Conference Proceedings (DAC '82)*, pages 175–181, Los Alamitos, Ca., USA, June 1982. IEEE Computer Society Press.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[20] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified *NP*-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[21] F. Glover. Tabu search–Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[22] M. T. Heath and P. Raghavan. A Cartesian parallel nested dissection algorithm. *SIAM Journal on Matrix Analysis and Applications*, 16(1):235–253, January 1995.

[23] B. Hendrickson and R. Leland. The chaco user's guide, version 2.0. Tech. Rep. SAND95-2344, Sandia National Labs, Albuquerque, NM, July 1995.

[24] D. S. Johnson, C. R. Aragon, L. A. Mcgeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Oper. Res.*, 37:865–892, 1989.

[25] G. Karypis, R.t Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proceedings of the 34th Conference on Design Automation (DAC-97)*, pages 526–529, NY, jun 1997. ACM Press.

[26] G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. In *Proceedings of Supercomputing'95*, San Diego, CA, December 1995. ACM/IEEE.

[27] G. Karypis and V. Kumar. hMETIS a hypergraph partitioning package version 1.5.3. Technical report, University of Minnesota, Department of Computer Science and Engineering, 1998.

[28] G. Karypis and V. Kumar. Multilevel k-way partioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[29] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, January 1999.

[30] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. *VLSI Design*, 11(3):285–300, 2000.

[31] B. W. Kernighan and S. Lin. An effective heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, pages 291–308, February 1970.

[32] S. Kirkpartick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, 13 May 1983*, 220(4598):671–680, 1983.

[33] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *J. Chem. Phys.*, 21(6):1087–1092, June 1953.

[34] G. L. Miller, S. Teng, W. Thurston, and S. A. Vavasis. Automatic mesh partitioning. In Alan George, John R. Gilbert, and Joseph W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, volume 56 of *IMA Volumes in Mathematics and its Applications*, pages 57–84. Springer, 1993.

[35] A. Pothen, H. D. Simon, and K. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, 11(3):430–452, July 1990. Sparse matrices (Gleneden Beach, OR, 1989).

[36] D. Ron. *Development of fast numerical solvers for problems in optimazation and statistical mechanics.* PhD thesis, The Weizmann Institute of Science.

[37] J. Ruge and K. Stüben. Algebraic multigrid. In S.F. McCormick, editor, *In Multigrid Methods*, pages 73–130. SIAM, 1987.

[38] I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge construction. *Elsevier Science*, 2003. To appear.

[39] E. Sharon, A. Brandt, and R. Basri. Fast multiscale image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR-00)*, pages 70–77, Los Alamitos, June 2000. IEEE.

[40] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Conf. Computer Vision and Pattern Recognition*, June 1997.

[41] H. D. Simon. Partitioning of unstructured problems for parallel processsing. *Computing Systems in Engineering*, 2(2/3):135–148, 1991.

[42] A. J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph Partitioning. (to appear in J. Global Optimization; originally published as Univ. Greenwich Tech. Rep. 00/IM/58), 2000.

[43] K. Stüben. An introduction to algebraic multigrid. In U. Trottenberg, C.W. Oosterlee, and A. Schüller, editors, *Appendix In Multigrid*, pages 413–532. Academic Press, 2001.

[44] K. Stüben. A review of algebraic multigrid. *J. Comput. Appl. Math.*, 128:281–309, 2001.

[45] C. Walshaw. http://staffweb.cms.gre.ac.uk/ c.walshaw/partition/.

[46] C. Walshaw. Multilevel refinement for combinatorial optimisation problems. Technical Report 01/IM/73, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, June 2001.

[47] C. Walshaw and M. Cross. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIJSSC: SIAM Journal on Scientific and Statistical Computing, apparently renamed SIAM Journal on Scientific Computing*, 22, 2000.

[48] S. Wishko-Stern. An algebraic multigrid based algorithm for bisectioning general graphs. M.Sc. Thesis, The Weizmann Institute of Science, 2004.