A New Algorithm for Multi-objective Graph Partitioning *

Kirk Schloegel, George Karypis, Vipin Kumar

[1] Department of Computer Science and Engineering,
University of Minnesota
[2] Army HPC Research Center,
Minneapolis, Minnesota

(kirk, karypis, kumar) @ cs.umn.edu

Abstract. Recently, a number of graph partitioning applications have emerged with additional requirements that the traditional graph partitioning model alone cannot effectively handle. One such class of problems is those in which multiple objectives, each of which can be modeled as a sum of weights of the edges of a graph, must be simultaneously optimized. This class of problems can be solved utilizing a multi-objective graph partitioning algorithm. We present a new formulation of the multi-objective graph partitioning problem and describe an algorithm that computes partitionings with respect to this formulation. We explain how this algorithm provides the user with a fine-tuned control of the tradeoffs among the objectives, results in predictable partitionings, and is able to handle both similar and dissimilar objectives. We show that this algorithm is better able to find a good tradeoff among the objectives than partitioning with respect to a single objective only. Finally, we show that by modifying the input preference vector, the multi-objective graph partitioning algorithm is able to gracefully tradeoff decreases in one objective for increases in the others.

1 Introduction

Graph partitioning is an important and well-understood problem. Here, the goal is to find a partitioning of a graph into k disjoint subdomains subject to the constraint that each subdomain has a roughly equal number of vertices, and with the objective to minimize the number of edges that are cut by the partitioning

* This work was supported by NSF CCR-9423082, by Army Research Office contracts DA/DAAG55-98-1-0441 and DA/DAAH04-95-1-0244, by Army High Performance Computing Research Center cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Additional support was provided by the IBM Partnership Award, and by the IBM SUR equipment grant. Access to computing facilities was provided by AHPCRC, Minnesota Supercomputer Institute. Related papers are available via WWW at URL: http://www-users.cs.umn.edu/~karypis

(referred to as the *edge-cut*). There are many applications for this problem from a wide range of diverse domains. Some examples are the parallelization of numerical simulations, computation of fill-reducing orderings of sparse matrices, the efficient fragmentation of databases, and the partitioning of VLSI circuits. The key characteristic of these applications is that they require the satisfaction of a single balance constraint along with the optimization of a single objective (i. e., the edge-cut).

Recently, a number of applications have emerged with additional requirements that the traditional (i.e., single-objective) graph partitioning model alone cannot effectively handle. Specifically, in many emerging applications there is a need to produce partitionings that try to optimize multiple objectives simultaneously. For example, a number of preconditioners have been developed that are focused on the subdomains assigned to each processor and ignore any intra-subdomain interactions (eg., block diagonal preconditioners and local ILU preconditioners). In these preconditioners, a block diagonal matrix is constructed by ignoring any intra-domain interactions, a preconditioner of each block is constructed independently, and then they are used to precondition the global linear system. The use of a graph partitioning algorithm to obtain the initial domain decomposition ensures that the number of non-zeros that are ignored in the preconditioning matrix is relatively small. However, the traditional partitioning problem does not allow us to control both the number as well as the magnitude of these ignored non-zeros. A partial solution to this problem can be obtained by assigning weights on the edges according to the magnitude of the corresponding non-zeros. Now, using traditional graph partitioning algorithms, we will obtain a decomposition that minimizes the sum of the magnitude of the non-zero entries that are ignored by the preconditioner. However, such an approach does not minimize the communication overhead incurred by parallel processing. This is because, this approach will prefer to put together vertices whose corresponding matrix value is large, at the expense of potentially *cutting* a large number of edges with small values. However, the communication overhead depends on the overall cut, and not on the magnitude of the corresponding edges. Ideally, we would like to obtain a decomposition that minimizes both the number of intra-domain interactions (reducing the communication overhead) and the numerical magnitude of these interactions (potentially leading to a better preconditioner).

Another example is the problem of minimizing the overall communications of parallel multi-phase computations. Multi-phase computations consist of m distinct computational phases, each separated by an explicit synchronization step. In general, the amount of interaction required between the elements of the mesh is different for each phase. Therefore, it is necessary to take the communications requirements of each phase into account in order to be able to accurately minimize the overall communications.

A natural extension to the graph partitioning problem is to assign a weight vector, w^e, of size m to each edge and a scalar weight, w^v, to each vertex. Now the k-way partitioning problem becomes that of partitioning the vertices of the graph into k disjoint subdomains such that each subdomain has a roughly equal amount of vertex weight and at the same time each one of the m objectives is optimized. Karypis and Kumar refer to this formulation as the *multi-objective graph partitioning problem* in [5].

Both the problems of minimizing the number and magnitude of the non-zero elements ignored by block-diagonal preconditioners and minimizing the overall communication of parallel multi-phase computations can be modeled as multi-objective partitioning problems. Figures 1 and 2 illustrate how this can be accomplished. In Figure 1(a), an eight-by-eight sparse (symmetric) matrix, M, is given. Figure 1(b) shows the corresponding 2-objective graph. Here, a vertex exists for each row of the matrix. An edge exists between vertices p and q if M_{pq} contains a non-zero element. The first value of the edge weight vector for every edge is one. The second value for an edge with incident vertices p and q is equal to M_{pq}. Figure 2 gives a 2-objective graph corresponding to a multi-phase computation. Here, the light vertices represent nodes that are active only during the first phase of the computation. The dark vertices represent nodes that are active during both the first and second phases of the computation. The edges are given edge weight vectors of either (1, 0) or (1, 1) depending on which phases their incident vertices are active. Note, that only those edges whose incident vertices are active during both of the phases are weighted (1, 1).

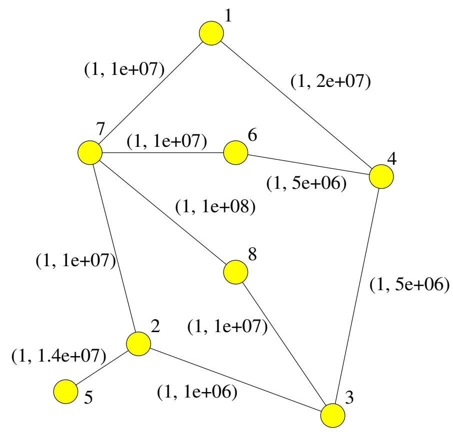
In this paper, we present a formulation for the multi-objective graph partitioning problem, as well as an algorithm for computing multi-objective partitionings with respect to this formulation. We explain how this scheme is able to be tuned by a user-supplied preference vector in order to control the tradeoffs among the different objectives in the computed partitionings, how this scheme results in predictable partitionings based on these inputs, and how this scheme is able to handle both similar and dissimilar objectives. We show that this multi-objective graph partitioner is better able to balance the tradeoffs of different objectives than partitioning with respect to a single objective only. We also show that by modifying the input preference vector, the multi-objective graph partitioning algorithm is able to gracefully tradeoff decreases in one objective for increases in the others.

2 Formulation of the Multi-objective Graph Partitioning Problem

One of the real difficulties in performing multi-objective optimization is that no single optimal solution exists. Instead, an optimal solution exists for each objective in the solution space. Furthermore, finding an optimal solution for one objective may require accepting a poor solution for the other objectives [7]. The result is that the definition of a good solution becomes ambiguous. This being the case, before a multi-objective graph partitioning algorithm can be developed, it is first necessary to develop a formulation that allows the user to disambiguate the definition of a good solution. This formulation should satisfy the following criteria.

	1	2	3	4	5	6	7	8
1	1e+07	--	--	2e+07	--	--	1e+07	--
2	--	1e+07	1e+06	--	1.4e+07	--	1e+07	--
3	--	1e+06	1e+07	5e+06	--	--	--	1e+07
4	2e+07	--	5e+06	1e+07	--	5e+06	--	--
5	--	1.4e+07	--	--	1e+07	--	--	--
6	--	--	--	5e+06	--	1e+07	1e+07	--
7	1e+07	1e+07	--	--	--	1e+07	1e+07	1e+08
8	--	--	1e+07	--	--	--	1e+08	1e+07

(a) Sparse Matrix



(b) Corresponding 2-Objective Graph

Fig. 1. A sparse symmetric matrix (a) and its corresponding 2-objective graph (b).

(a) It should allow fine-tuned control of the tradeoffs among the objectives. That is, the user should be able to precisely control the amount that one or more objectives may be increased in order to decrease the other objectives when these objectives are in conflict with one another.

(b) The produced partitionings should be predictable and intuitive based on the user's inputs. That is, the output partitioning should correspond to the user's notion of how the tradeoffs should interact with each other.

(c) The formulation should be able to handle objectives that correspond to quantities that are both of similar as well as of different types. Consider the example of minimizing the overall communications of a multi-phase computation. Here, all of the objectives represent similar quantities (i. e., communications overhead). However, other applications exist whose objectives are quite dissimilar in nature. The preconditioner application is an example. Here, both the number and the magnitude of the non-zero elements off of the diagonal are to be minimized. Minimizing the number of non-zero elements off of the diagonal will reduce the communications required per iteration of the computation, while minimizing the magnitude of these elements will improve its convergence rate. These are quite dissimilar in nature.

Two straightforward means of disambiguating the definition of a good multi-objective solution are (i) to prioritize the objectives, and (ii) to combine the objectives into a single objective. We next discuss each of these in the context of the above formulation criteria.

Priority-based Formulation. The definition of a good multi-objective solution is ambiguous when the relationship between the objectives is unclear. One of the most straightforward means of defining this relationship is to list the objectives in order of priority [2, 9]. Therefore, one possible formulation of the

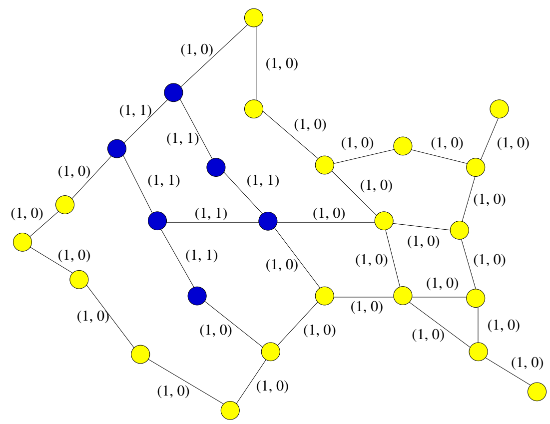


Fig. 2. This is an example of a 2-phase computation. The light vertices are active only during the first phase of the computation. The dark vertices are active during both phases. The edge weight vectors indicate the phases in which the edges are active.

multi-objective graph partitioning problem is to allow the user to assign a priority ranging from one to m to each of the m objectives. Now, the multi-objective partitioning problem becomes that of computing a k-way partitioning such that it simultaneously optimizes all m objectives, giving preference to the objectives with higher priorities.

This formulation is able to handle objectives of different types as the relationship between the objectives is well-described and the objectives are handled separately. It will also result in predictable partitionings, in that the highest priority objective will be minimized with the rest of the objectives minimized to the amount possible without increasing higher priority objectives. However, this formulation provides the user with only a coarse-grain control of the tradeoffs between the objectives. For example, consider a two-objective graph from which three partitionings are computed. The first has an edge-cut vector of $(1.0, 100.0)$, the second has an edge-cut vector of $(1.01, 2.0)$, and the third has an edge-cut vector of $(500.0, 1.0)$. Typically, the second of these would be preferred, since compared to the first partitioning, a 1% increase in the edge-cut of the first objective will result in a 98% decrease in the edge-cut of the second objective. Also, compared to the third partitioning, a 100% increase in the second objective will yield a 99.8% decrease in the first objective. However, under the priority-based formulation, the user is unable to supply a priority list that will produce the second partitioning. This is because partitionings exists with better edge-cuts for each of the two objectives. From this example, we see that the priority-based formulation is unable to allow the user fine-tuned control of the tradeoffs between the objectives.

Combination-based Formulation. While prioritizing the objectives describes their relationship well enough to disambiguate the definition of a good solution, it is not powerful enough to allow the user fine-tuned control over the tradeoffs among the objectives. One simple approach that has been used in other domains is to combine the multiple objectives into a single objective and then to use a single objective optimization technique [1, 6, 9]. Typically, this is done by taking the sum of the elements of the objective vector weighted by a *preference* vector, p. A formulation of the multi-objective graph partitioning problem that uses this approach is to assign to each edge a scalar combined weight, w^c, such that

$$w^c = \sum_{i=1}^{m} w_i^e p_i. \tag{1}$$

The resulting graph can then be partitioned utilizing an existing (single-objective) graph partitioning algorithm. For example, if we would like to scalarize the edge weights of a three-objective problem and want to give the second objective five times the weight of the first and third objectives, we could use a preference vector of (1, 5, 1). An edge with a weight vector of (2, 2, 1), for example, would then be assigned a combined weight of 2+10+1=13.

Unlike the priority-based formulation, this one allows a fine-tuned control of the tradeoffs among the objectives, since the objectives are weighted and not merely ordered. However, this formulation cannot handle dissimilar objectives. This is because it requires that the objectives be combined (by means of a weighted sum). For dissimilar objectives (such as the number and magnitude of the ignored elements off of the diagonal), this combination can be meaningless.

One possible solution is to divide each edge weight by the average edge weight for the corresponding objective in an attempt to normalize all of the objectives. Normalization will help us to combine dissimilar objectives in a more meaningful way. However, this solution fails the predictability criteria. Consider the example two-edge-weight graph depicted in Figure 3(a). In the center of this graph is a clique composed of edges with edge weight vectors of (10000, 1), while the rest of the edges have vectors of (1, 1). (Note, not all of the edges have their weights marked in the figure.) In this example, the first edge weight represents the degree to which we would like to have the vertices incident on an edge to be in the same subdomain. The second edge weight represents the interaction between vertices as normal. Therefore, in partitioning this graph, we have two objectives. (i) We would like the vertices of the clique to be in the same subdomain. (ii) We would like to minimize the edge-cut. The intuitive meaning of this graph is that the clique should be split up only if doing so reduces the edge-cut by tens of thousands. Figure 3(b) gives the new edge weights after normalization by the average edge weight of each objective. Here we see that the first edge weights of the clique edges have been scaled down to about five, the first edge weights of the non-clique edges have been scaled down to about zero, and the second edge weights of all edges have remained at one. Now, consider what will happen if we input different preference vectors and partition the graph. If the preference vector gives each objective equal weight (i.e., a preference vector of (1, 1)), then the clique will not be split during partitioning. This is as expected, since the clique edges in the input graph have very high edge weights for the first objective. Also,

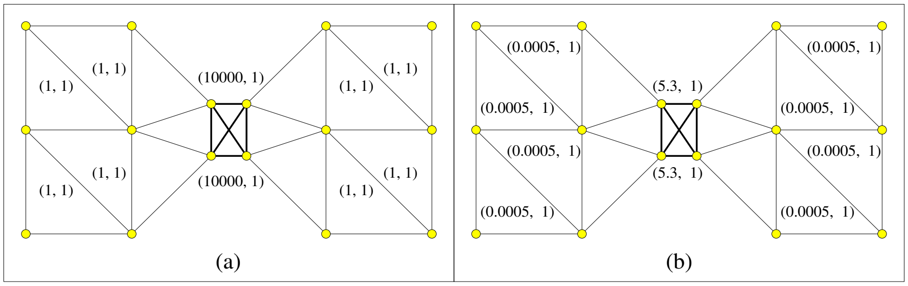


Fig. 3. This is an example of a two-edge-weight graph. The edges of the clique in the center of the graph are weighted (10000, 1) in (a) and (5.3, 1) in (b). The other edges are weighted (1, 1) in (a) and (.0005, 1) in (b). The edge weights in (b) are computed by normalizing the edge weights in (a) by the average edge weights of the corresponding objective.

if we favor the first edge weight to any extent (i.e., supply a preference vector between (2, 1) and (1000000, 1)), this trend continues. However, if we favor the second edge weight only moderately (eg., a preference vector of (1, 6)), then the optimal partitioning will split the clique. This is because normalizing by the average edge weight has caused the first edge weights of the clique edges to be scaled down considerably compared to the second edge weights. The result is that we lose the intuitive meaning that the first edge weights had in Figure 3(a).

3 A Multi-objective Graph Partitioning Algorithm

As discussed in the previous section, existing formulations of multi-objective optimization problems, including that of multi-objective graph partitioning, do not fully address the requirements of the multi-objective graph partitioning problem. In this section, we present a novel new formulation that allows the user to control of the tradeoffs among the different objectives, produces predictable partitionings, and can handle objectives of both similar as well as dissimilar type.

Our formulation is based on the intuitive notion of what constitutes a good multi-objective solution. Quite often, a natural way of evaluating the quality of a multi-objective solution is to look at how close it is to the optimal solutions for each individual objective. For example, consider a graph with two objectives, let $P_{1,2}$ be a multi-objective partitioning for this graph, and let C_1 and C_2 be the edge-cuts induced by this partitioning for the first and second objectives, respectively. Also, let P_1 be the optimal partitioning of the graph with respect to only the first objective, and let C_1^o be the corresponding edge-cut. Finally, let P_2 be the optimal partitioning of the graph with respect to only the second objective, and let C_2^o be the corresponding edge-cut. Given these definitions, we can easily determine whether or not $P_{1,2}$ is a good multi-objective partitioning by comparing C_1 against C_1^o and C_2 against C_2^o. In particular, if C_1 is very close to C_1^o and C_2 is very close to C_2^o, then the multi-objective partitioning is very good. In general, if the ratios C_1/C_1^o and C_2/C_2^o are both close to one, then the solution is considered to be good.

Using this intuitive notion of the quality of a multi-objective partitioning, we can define a scalar combined edge-cut metric, c^c, to be equal to

$$c^c = \sum_{i=1}^{m} \frac{c_i}{c_i^o} \tag{2}$$

where c_i is equal to the actual edge-cut of the ith objective, and c_i^o is equal to the optimal edge-cut of the ith objective. We can augment this definition with the inclusion of a preference vector, p. So Equation 2 becomes

$$c^c = \sum_{i=1}^{l} p_i \frac{c_i}{c_i^o}. \tag{3}$$

Equation 3 then becomes our single optimization objective. In essence, minimizing this metric attempts to compute a k-way partitioning such that the edge-cuts with respect to each objective are not far away from the optimal edge-cuts. The distance that each edge-cut is allowed to stray from the optimal is determined by the preference vector. A preference vector of (1, 5) for example, indicates that we need to move at least five units closer to the optimal edge-cut of the second objective for each one unit we move away from the optimal edge-cut of the first objective. Conversely, we can move one unit closer to the optimal edge-cut of the first objective if it moves us away from the optimal edge-cut of the second objective by five units or less. In this way, the preference vector can be used to traverse the area between the optimal solutions points of each objective. This results in predictable partitionings based on the user-suppled preference vector as well as fined-tuned control of the tradeoffs among the objectives. Finally, since dividing by the optimal edge-cuts in Equation 3 normalizes the objectives, we can combine similar and dissimilar objectives in this way.

If we further expand the c_i term of Equation 3, it becomes

$$= \sum_{i=1}^{m} p_i \frac{\sum_{j \epsilon cut} w_i^{e_j}}{c_i^o} \tag{4}$$

where the term $\sum_{j \epsilon cut} w_i^{e_j}$ represents the sum of the ith weights of the edges cut by the partitioning. By manipulating this equation, we get the following.

$$= \sum_{j \epsilon cut} \left(\sum_{i=1}^{m} \frac{p_i w_i^{e_j}}{c_i^o} \right) \tag{5}$$

The term $\sum_{i=1}^{m} \frac{p_i w_i^{e_j}}{c_i^o}$ in Equation 5 gives the scalar combined weight of an edge (defined in Equation 1) with each weight normalized by the optimal edge-cut of the corresponding objective. Since Equations 3 and 5 are equal, we have

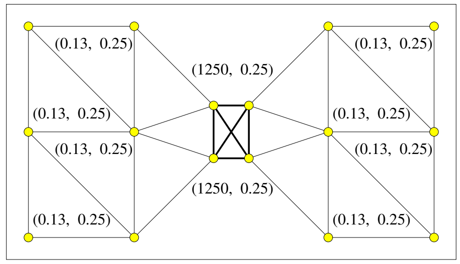


Fig. 4. This shows the resulting two-edge-weight graph constructed by normalizing each edge weight of the graph in Figure 3(a) by the optimal edge-cut for the corresponding objective.

shown that minimizing the normalized combined weights of the edges cut by the partitioning minimizes the combined edge-cut metric, c^c. That is, the problem of computing a k-way partitioning that optimizes Equation 3 is identical to solving a single-objective partitioning problem with this proper assignment of edge weights.

Consider what happens if we apply this technique to the problem in Figure 3(a). Figure 4 gives new edge weights after the edge weights from the graph in Figure 3(a) are normalized by the optimal edge-cut of the corresponding objective. Here, we see that the first edge weights of the clique edges are still quite high compared to the second edge weights. Therefore, we must favor the second edge weights by a great amount (eg., (1, 5000)) before the clique is split. Therefore, we see how the scheme of normalizing by the optimal edge-cut obtains more predictable results given a user-supplied preference vector than normalizing by the average edge weights of an objective.

We have developed a multi-objective graph partitioning algorithm that is able to compute partitionings with respect to this new formulation. In our algorithm, we first utilize the k-way graph partitioning algorithm implemented in METIS [4] to compute a partitioning for each of the m objectives separately. We record the best edge-cuts obtained for each objective. Next, our scheme assigns to each edge a combined weight equal to the sum of the each edge weight vector normalized by the best edge-cuts and weighted by the preference vector. In the final step, we again utilize METIS to compute a partitioning with respect to these new combined edge weights.

4 Experimental Results

The experiments in this section were performed using the graph, MDUAL2. It is a dual of a 3D finite element mesh and has 988,605 vertices and 1,947,069 edges. We used this graph to construct two types of 2- and 4-objective graphs. For the first type, we randomly select an integer between one and 100 for each edge weight element. For the second type, we obtained the edge weights in the

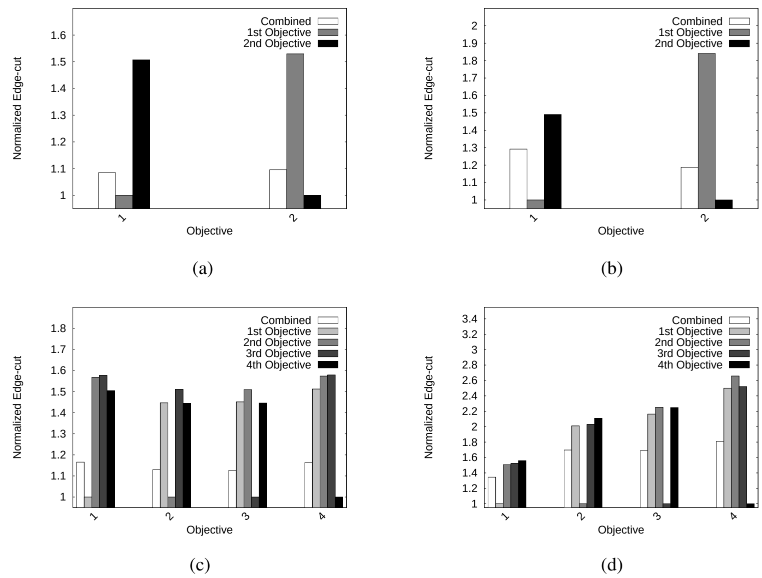


Fig. 5. This figure compares the normalized edge-cut results obtained by the multi-objective graph partitioner and those obtained by partitioning with respect to each of the objectives alone for a 2-objective Type 1 problem (a), a 2-objective Type 2 problem (b), a 4-objective Type 1 problem (c), and a 4-objective Type 2 problem (d).

following manner. For the first objective, we computed a set of nine different 7-way partitionings utilizing the k-way graph partitioning algorithm in METIS and kept a record of all of the edges that were ever cut by any one of these partitionings. Then, we set the first edge weight element of each edge to be one if this edge had ever been cut, and five otherwise. Next, we computed a set of eight different 11-way partitionings utilizing the k-way graph partitioning algorithm in METIS and set the second edge weight elements to one if the edge had been cut by any one of these partitionings and 15 otherwise. For the 4-objective graphs we then similarly computed a set of seven different 13-way partitionings and set the third edge weight elements to either one or 45 depending on whether the corresponding edge had been cut. Finally for the 4-objective graphs, we computed a set of six 17-way partitionings and similarly set the fourth edge weight elements to either one or 135.

We generated the Type 1 problems to evaluate our multi-objective graph partitioning algorithm on randomly generated problems. We generated the Type 2 problems to evaluate our algorithm on some particularly difficult problems. That is, each graph should have a small number of good partitionings for each objective. However, our strategy of basing these good partitionings on 7-, 11-, 13-, and 17-way partitionings was designed to help ensure that the good partitionings of each objective do not significantly overlap. Therefore, computing a single partitioning that simultaneously minimizes each of these objectives is difficult.

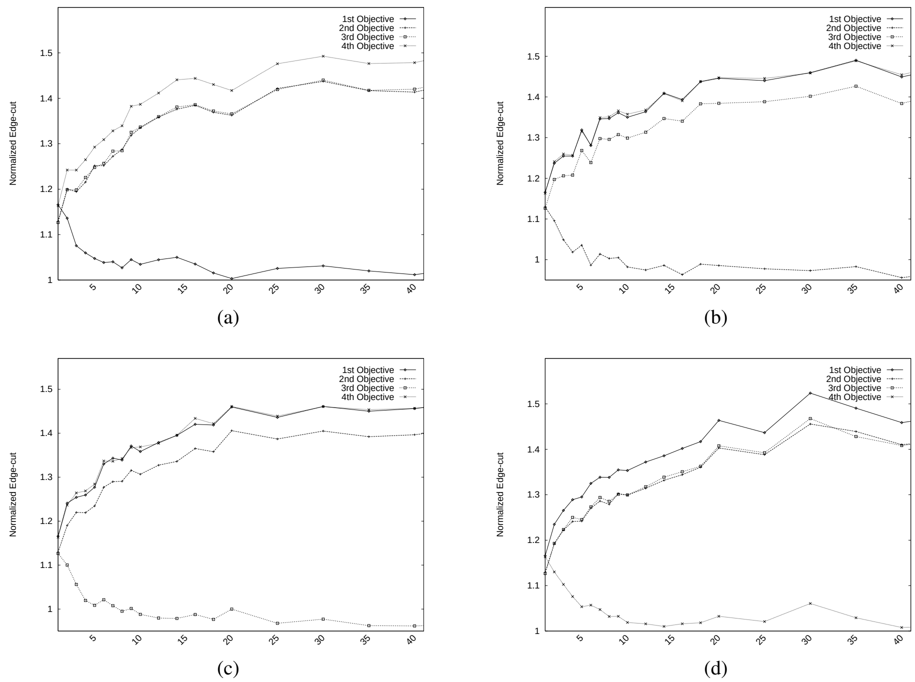


Fig. 6. This figure gives the normalized edge-cut results for 4-objective 64-way partitionings of Type 1 problems. The preference vector used in (a) is (x, 1, 1, 1), (b) is (1, x, 1, 1), (c) is (1, 1, x, 1), and (d) is (1, 1, 1, x).

Qualitative Evaluation of the Multi-objective Graph Partitioner. Figure 5 compares the results obtained by the multi-objective graph partitioner with those obtained by partitioning with respect to each of the objectives alone with the single objective graph partitioner implement in MeTiS. Specifically, we show the results of 2- and 4-objective 64-way partitionings of Type 1 and 2 problems. In Figure 5(a) we give the results of partitioning the 2-objective Type 1 problem. In (b) we give the results of partitioning the 2-objective Type 2 problem. In (c) we give the results of partitioning the 4-objective Type 1 problem. Finally, in (d) we give the results of partitioning the 4-objective Type 2 problem. All of the edge-cuts are normalized by those obtained by partitioning with respect to a single objective only. Therefore, they give an indication of how far away is each objective from the optimal edge-cut.

Figure 5 shows that our multi-objective algorithm is able to compute partitionings such that a good tradeoff is found among the edge-cuts of all of the objectives. Partitioning with respect to a single objective obtains good edge-cut results for only a single objective. All of the other objectives are worse than those obtained by the multi-objective algorithm.

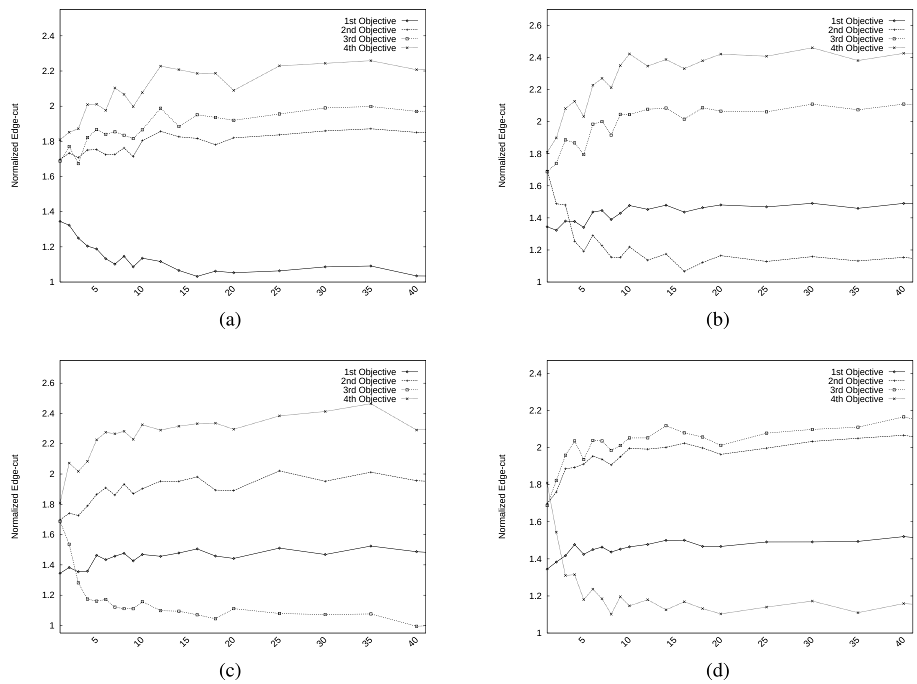


Fig. 7. This figure gives the normalized edge-cut results for 4-objective 64-way partitionings of Type 2 problems. The preference vector used in (a) is (x, 1, 1, 1), (b) is (1, x, 1, 1), (c) is (1, 1, x, 1), and (d) is (1, 1, 1, x).

Control of the Tradeoffs by the Preference Vector. Figures 6 and 7 demonstrate the ability of our multi-objective graph partitioner to allow fine-tuned control of the tradeoffs of the objectives given a user-suppled preference vector. Specifically, these give the results of a number of preference vectors for 4-objective 64-way partitionings of Type 1 and 2 problems. Here, three of the four elements of the preference vector are set to one, while the fourth is set to a value x. This value is plotted on the x-coordinate. The y-coordinate gives the values of the edge-cuts of each of the four objectives obtained by our multi-objective algorithm. So, as we move in the direction of positive infinity on the x-axis, a single objective is increasingly favored. Figure 6 gives the results obtained on Type 1 problems. Figure 7 gives the results obtained on Type 2 problems. In both figures, the preference vector used in (a) is (x, 1, 1, 1), (b) is (1, x, 1, 1), (c) is (1, 1, x, 1), and (d) is (1, 1, 1, x). All of the edge-cuts are normalized by those obtained by partitioning with respect to a single objective only.

Figures 6 and 7 show that by increasing the values of one of the elements of the preference vector, it is possible to gracefully tradeoff one objective for the others with the multi-objective partitioner. We see that in each result, the value at $x = 1$ is a good tradeoff among the four objectives. As x is increased, the edge-cut of the corresponding objective approaches that of the partitioning with respect to that objective only. The edge-cuts of the other objectives increase gracefully.

5 Conclusion

We have described a new formulation for the multi-objective graph partitioning problem and an algorithm that computes multi-objective partitionings with respect to this formulation. We have shown that this algorithm is able to minimize the edge-cut of a multi-objective problem given a user-supplied preference vector. We have shown that this algorithm provides the user with a fine-tuned control of the tradeoffs among the objectives, results in intuitively predictable partitionings, and is able to handle both similar and dissimilar objectives. Finally, we have shown that this algorithm is better able to find a good tradeoff between the objectives than partitioning with respect to a single objective only.

Parallelizing the multi-objective graph partitioning algorithm is quite trivial, as highly scalable implementations of parallel graph partitioners exist [3, 8]. These codes assume that the graph is distributed across the processors. They return the edge-cut results to each of the processors. Thus, the parallel formulation consists of a number of calls to a parallel graph partitioning routine equal to the number of objectives followed by a phase in which each processor independently computes its vertices' normalized combined edge weights. Finally, one last call to the parallel partitioner utilizing these new edge weights will produce the multi-objective partitioning.

References

1. P. Fishburn. *Decision and Value Theory.* J. Wiley & Sons, New York, 1964.
2. Y. Haimes and W. Hall. Multiobjectives in water resource systems analysis: The surrogate trade-off method. *Water Resources Research*, 10:615–624, 1974.
3. G. Karypis and V. Kumar. A parallel algorithms for multilevel graph partitioning and sparse matrix ordering. Technical Report TR 95-036, Department of Computer Science, University of Minnesota, 1995. Also available on WWW at URL http://www.cs.umn.edu/˜karypis. A short version appears in Intl. Parallel Processing Symposium 1996.
4. G. Karypis and V. Kumar. METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL *http://www.cs.umn.edu/˜metis.*
5. G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. Technical Report TR 98-019, Department of Computer Science, University of Minnesota, 1998.
6. R. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs.* J. Wiley & Sons, New York, 1976.
7. M. Makowski. Methodology and a modular tool for multiple criteria analysis of lp models. Technical Report WP-94-102, International Institute for Applied Systems Analysis, 1994.

8. C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 47(2):102–108, 1997.
9. P. Yu. *Multiple-Criteria Decision Making: Concepts, Techniques, and Extensions*. Plenum Press, New York, 1985.