UCLA
COMPUTATIONAL AND APPLIED MATHEMATICS

An Efficient Low Cost Greedy Graph Partitioning Heuristic

P. Ciarlet, Jr.
F. Lamour

January 1994

CAM Report 94-1

Department of Mathematics
University of California, Los Angeles
Los Angeles, CA. 90024-1555

An Efficient Low Cost Greedy Graph Partitioning Heuristic

P. Ciarlet, Jr * F. Lamour †

January 11, 1994

Abstract

The problem considered in this report is the graph partitioning problem. This problem consists of splitting the set of nodes of a given graph into balanced connected subsets. Moreover the overall number of edges which have their ends in different subsets must be minimized. We propose a very fast heuristic to approximate the solution of this problem for graphs derived from numerical problems.

1 Introduction

Many problems can be represented by graphs where nodes stand for distinct amount of work and edges between nodes schematize the information exchanges.

Among the problems that one can encounter is how to decompose well the entire graph problem in several subgraphs in order to solve smaller problems simultaneously, without having to communicate too much between different subgraphs.

This is identified as the graph partitioning problem (GP-problem for short). More formally the problem is written as follows. Let $G = (V, E)$ be a graph where V is the set of nodes and E is the set of edges. The question is how to partition V into p subsets V_1, V_2, ..., V_p such that:

- $V_i \bigcap V_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^{p} V_i = V$,
- $\mid V_i \mid = \mid V_j \mid$ for $1 \leq i, j \leq p$ and $i \neq j$,
- $\mid \{(v_i, v_j) \in E \; and \; v_i \in V_i, \; v_j \in V_j \; with \; i \neq j\} \mid$ is as small as possible.

*Department of Mathematics, Univ. of Calif. at Los Angeles, CA 90024 and Commissariat à l'Energie Atomique, CEL-V, D.MA, MCN, 94195 Villeneuve-St-Georges Cedex, France. E-mail: ciarlet@math.ucla.edu, ciarlet@etca.fr. The work of this author was partially supported by the DGA/DRET under contract 93-1192.

†Department of Mathematics, Univ. of Calif. at Los Angeles, CA 90024. E-mail: lamour@math.ucla.edu. The work of this author was partially supported by the ARO under contract DAAL03-91-C-0047 (Univ. of Tennessee) and by the ONR under contract N00014-92-J-1890.

Unfortunately, the GP-problem is an NP-hard problem and so we must be content to find good heuristics.

Here our concern is to approximate the GP-problem rapidly for graphs which come from meshes and finite element discretization. Briefly there are two ways of looking at that problem. One is based on spectral properties of the matrix associated to the graph. Among the different works, let us mention those of [6], [5] and [4]. The other way is to work directly on the graph by applying greedy methods (see [2] and [3]). Our approach is from the second school.

This report resumes the outline of our work, already described in [1], and completes it. It is organized as follows. Section 2 gives a general definition of greedy algorithms according to the GP-problem and evaluates the complexity of such algorithms. Section 3 presents the main steps of our graph partitioning algorithm called PAM with some intuitive justifications of our choices. The overall complexity of our algorithm is also addressed in this section. The implementation aspects of the algorithm are explained in section 4. We present a number of representative examples in section 5 and finally conclude in the last section.

2 About greedy algorithms

Greedy algorithms are a natural and naive way to look at some problems. In some cases, it seems to give reasonable results, especially here for approximating the solution of the GP-problem.

According to the GP-problem, a greedy algorithm can be described as an algorithm that computes one after another each subset V_i by simply accumulating nodes when traveling through the graph. The only problematical questions are: how to start and how to stop?

The way of accumulating nodes in each subset is obvious from the graph structure of the problem. A starting node v_s is chosen and marked. The accretion process is done by selecting and marking the neighbors of v_s, then the unmarked neighbors of the neighbors of v_s and so on as long as the expected total number of nodes is not reached. This can be viewed as successively building fronts.

The way of choosing a starting node v_s will clearly affect the shape of the final partition. It will also influence the communication scheme, i.e. the number of existing edges between different subsets of the partition.

In the same way, the manner that one chooses the prescribed number of nodes among all the candidate nodes of the last front contributes to the quality of the final partition.

Thus a greedy heuristic for solving the GP-problem can be defined roughly by iterating the following 3 steps:

1. Choose a "good" starting node v_s,
2. Accumulate enough descendants of v_s,
3. Stop according to some tie-break strategy in case of multiple choices.

At present, there are no theoretical results on the "goodness" of one starting node. Neither are there results on how good a tie-break strategy is. For those two points only intuitive guesses help to design GP-problem heuristics. However, an obvious justification of using greedy heuristics for solving the GP-problem is that they are inexpensive.

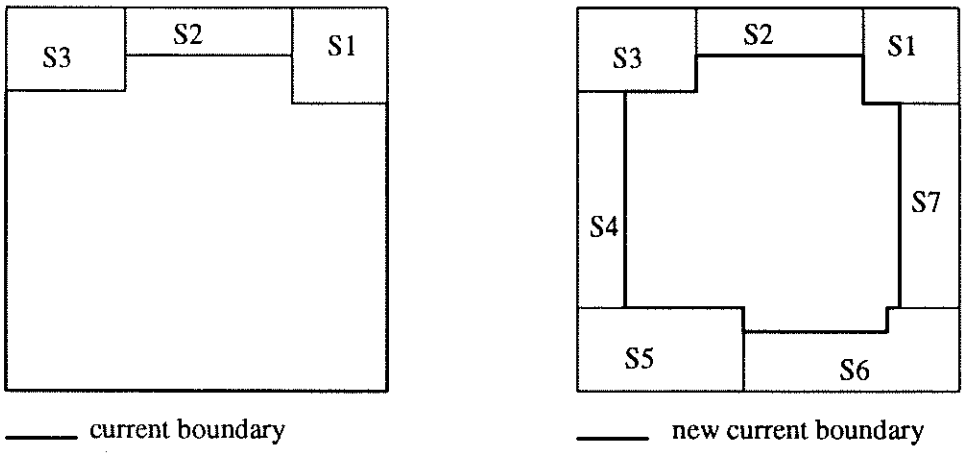

Figure 1: Current boundaries

Indeed, let us find out the overall complexity of such algorithms. Let $G = (V, E)$ be a graph such that $N = \mid V \mid$ and $M = \mid E \mid$. More, let d denote the average degree of a node in G, and d_{max} the maximum degree of a node in G. Then the complexity of dividing V into p subsets each having $n = N/p$ nodes can be derived by estimating the cost of each step of iteration i of the partitioning process.

Step 1. Choose a good starting node among the remaining nodes in $O(N)$ operations.

Step 2. Accumulate iteratively the neighbors of the previous chosen nodes in $O(n.d)$ operations.

Step 3. This step amounts to sorting the nodes of the last front according to some criteria. The number of nodes of the last front can be as large as $O(n)$. Thus it can be done in $O(n \log_2(n))$ using a well-known sorting.

So for the general case the overall complexity is $\max(p, d, \log_2(\frac{N}{p})).O(N)$.

3 PAM a greedy partioning heuristic

We now present PAM, our greedy partitioning algorithm, by defining precisely the 3 steps enumerated in the previous section.

In our case, our general purpose is to solve the GP-problem for graphs that come from physical meshes, usually two- or three-dimensional. Thus we can talk about the boundary of the graph. We call n_b the number of boundary nodes.

As greedy algorithms do, PAM builds iteratively the different subsets of the partition by accumulating nodes in each subset and marking them when they have been selected. So at each iteration one can define the *current boundary* as the set of nodes that belong to the boundary of the graph and that have not been marked yet. We can extend the notion of current boundary even when there is no more unmarked boundary node, by defining a new one and updating n_b. In that case the new current boundary is the set of unmarked nodes

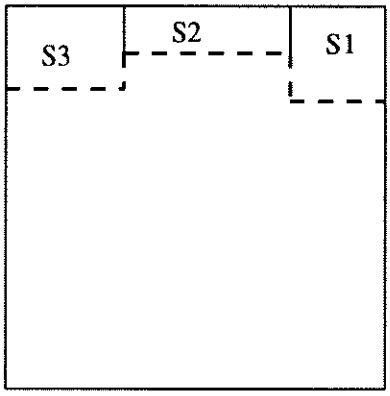

Figure 2: Virtual boundary

that are the neighbors of marked nodes. Figure 1 illustrates the different definitions of the current boundary.

A way to redefine the current boundary at low cost is to introduce the notion of *virtual boundary*. Actually, this virtual boundary is defined, at each iteration, as the set of unmarked nodes that are neighbors of marked nodes, excluding current boundary nodes. Then, the new current boundary is defined as the virtual boundary and the new virtual boundary is build anew. See Figure 2 for an example of virtual boundary.

Finally, let us introduce the notion of the *updated degree* of a node: it is simply the number of unmarked neighbors of this node. Note that the updated degree starts from the degree of the node and decreases to zero as the partitioning algorithm goes along.

Now, let us recall that N is the total number of nodes and that $n = N/p$ is the expected number of nodes by subsets for a partition into p subsets. Then the whole process is described as follows.

While $i < p$

1. Choose an unmarked node v_s such that:
   (a) v_s belongs to the current boundary,
   (b) if the current boundary is not new, v_s is a neighbor of a node belonging to V_{i-1} (if possible[1]),
   (c) v_s has a minimal updated degree.

   Mark v_s and initialise V_i with v_s.
2. Select the unmarked neighbors of nodes in V_i, and let k be their number.
   If $\mid V_i \mid + k < n$ then mark those nodes, add them to V_i and update the updated degree of their neighbors, and then return to 2.
3. Mark $(n - \mid V_i \mid)$ minimal updated degree nodes and add them to V_i.
   Update the current and virtual boundaries.
   Do $i = i + 1$.

[1]It may not be possible to find a node neighboring the previously built subset if the boundary is multiconnected.

Mark all the remaining nodes and add them into V_p.

Let us discuss about the intuitive justifications of our starting node as well as our tie-break strategies.

By choosing a node on the current boundary that verifies condition (b), one can be convinced that this will provide the overall regularity of the partition. As a matter of fact, because of the definition of the current boundary, the subsets will be built around the boundary in a concentric way. Condition (c) goes in the sense of the third requirement of the GP-problem, i.e. tries to reduce the number of edges which have their endpoints in different subsets. Intuitively, we can explain this by saying that the number of interface nodes (nodes which have neighbors in other subsets) will be dependent on the number of neighbors of the starting node. Choosing a starting node with a minimal updated degree consists of exploring the nodes more in depth first search than in breath first search.

The tie-break strategy in step 3, which dictates the selection of minimal updated degree nodes, also ensures the minimization of the number of inter-subset edges. Let us now estimate the complexity of the PAM algorithm.

Step 1. In general, this is done in $O(n.d)$ operations. Indeed, we look at neighbor nodes of the previous subset satisfying criteria (a), (b) and (c) in $O(n.d)$ operations. Otherwise, we choose a node satisfying (a) and (c) in $O(n_b)$ operations. This happens only once for each current boundary if the boundary is simply connected. If the boundary is multiconnected (q components), then it can happen q times for the original boundary.

Step 2. As for the general algorithm, it requires $O(n.d)$ operations.

Step 3. First, the number of nodes of the last front is less than n. As the updated degree of a node varies between 0 and d_{max}, sorting them is done in $O(n)$ operations by piling them up in $(d_{max}+1)$ heaps. On the other hand, updating the current and virtual boundaries is done respectively in $O(n)$ and $O(n.d)$ operations.

It is easily seen that a node cannot belong to two different current boundaries. Therefore, $\sum n_b \leq N$. This proves that the overall complexity our partitioning algorithm is $d.O(N)$ (or $O(M)$).

4 Implementation aspects

4.1 Connected subsets

One important property that we have not talked about is the connectivity of the subsets of the partition. It could happen, in our algorithm, during the construction of one subset, that there are no more unmarked nodes while the subset in progress has not reached its final size. One way to complete the subset is to choose a new starting node and to start again the process until the number of expected nodes is attained. That would lead to a multiconnected subset. To avoid that we reassign the nodes of the incomplete subset to the neighboring subsets following the rule that a node is assigned to the subset where most of its neighbors are. Then we rebuild the current subset.

In the algorithm presented on pages 4-5, the last subset is trivialy built by choosing all the remaining nodes. Once again the connectivity of the subset is not guaranteed. In fact, we have implemented differently the construction of the last subset so that it remains connected in most cases. In the case of multiple components of the last subset, we reassign the nodes of the smallest ones (i.e. those which are less then 20 % of the total size) to neighboring subsets following the same rule described previously. When two or more components are of the same size (greater than 20 % of the total size), which by experience do not comes often, we accept the fact that the last subset is not connected.

The complexity of our algorithm does not deteriorate when these extra steps are performed. First, reassigning nodes of an incomplete subset amounts to a backward step 2. Second, its reconstruction correspond to one more iteration (steps 1, 2 and 3). As for the special treatment of the last subset, it is actually performed inside the usual iteration plus, possibly, one or more reassignment and reconstruction steps.

4.2 Balanced subsets

We have called n the number of expected nodes by subset. This number is defined as the total number of nodes in the graph, N, divided by the number of expected subsets, p. As the division of N by p is not necessarily exact and as we reassign nodes to some subsets already built, it could happen that the subsets are not balanced. To prevent this phenomenom we update the value of expected nodes called n_i before each new construction of a subset. Then at each iteration n_i is updated as:

$$n_i = \frac{N - \sum_{j=1}^{i-1} n_j}{p - (i-1)}.$$

Notice that updating the value of the number of nodes by subset does not modify the complexity of our partitioning algorithm.

5 Examples

We present here 6 graph examples which are well known to be good tests for the robustness of heuristic partitioning methods. The first five examples are commonly known graphs such as an Eppstein graph, a Spiral graph, a Parc graph, an Airfoil graph and a Barth graph. The last one is the nine-point finite differences discretization of a square.

Let us first introduce a definition. If G has been partitioned into p subgraphs then we call G_{ind} the induced graph such that $\{1, 2, ..., p\}$ is its nodes set and that there exists an edge between node i and node j when there exists at least one edge between a node in V_i and one in V_j.

We summarize in a table, for all examples, our partitioning results relatively to 6 parameters: the number of subsets (p), the average number of nodes by subset (n_{av}), the percentage of inter-subset edges (e_{av}), the average degree of the induced graph ($d_{av}(G_{ind})$), the number of connected components of the last subset (cc), and finally the elapsed time[2] in second (T_p). Our examples were run on a Sun SparcStation 10/30.

[2] it is equal to the difference of the wall-clock times read after and before the execution of the PAM subroutine.

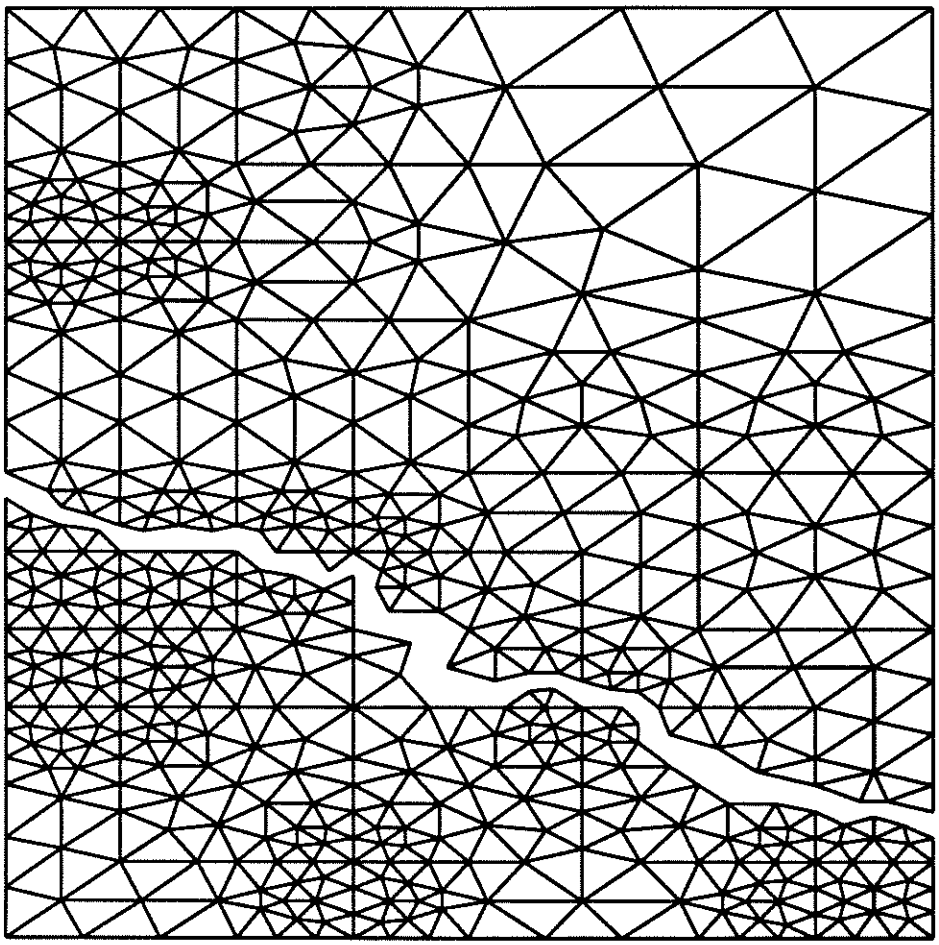

Figure 3: Eppstein graph partitioned into 2 subsets

p	n_{av}	e_{av} (%)	$d_{av}(G_{ind})$	cc	T_p (s)
2	274	4	1	1	0.01
8	68	14	3	1	0.01
15	36	20	4	1	0.01

Table 1: Eppstein with 547 nodes and 1566 edges.

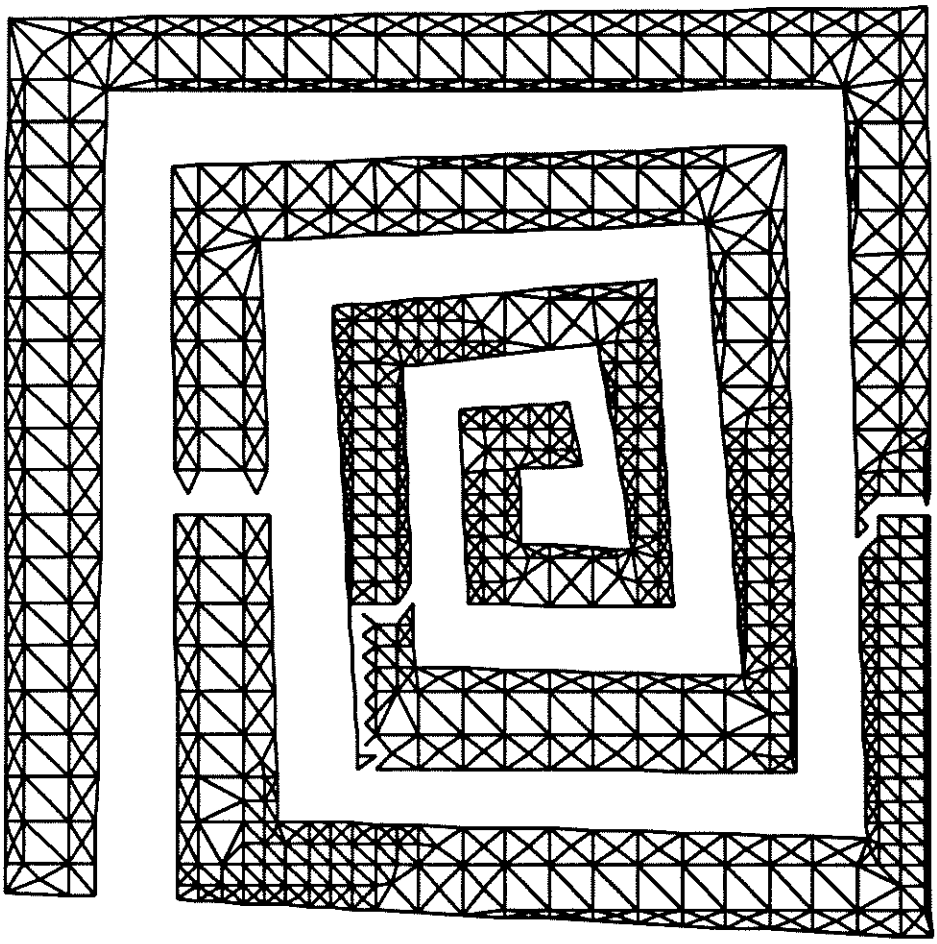

Figure 4: Spiral graph partitioned into 4 subsets

p	n_{av}	e_{av} (%)	$d_{av}(G_{ind})$	cc	T_p (s)
4	300	1	2	1	0.02
9	133	2	2	1	0.02
25	48	9	2	1	0.02

Table 2: Spiral with 1198 nodes and 3189 edges.

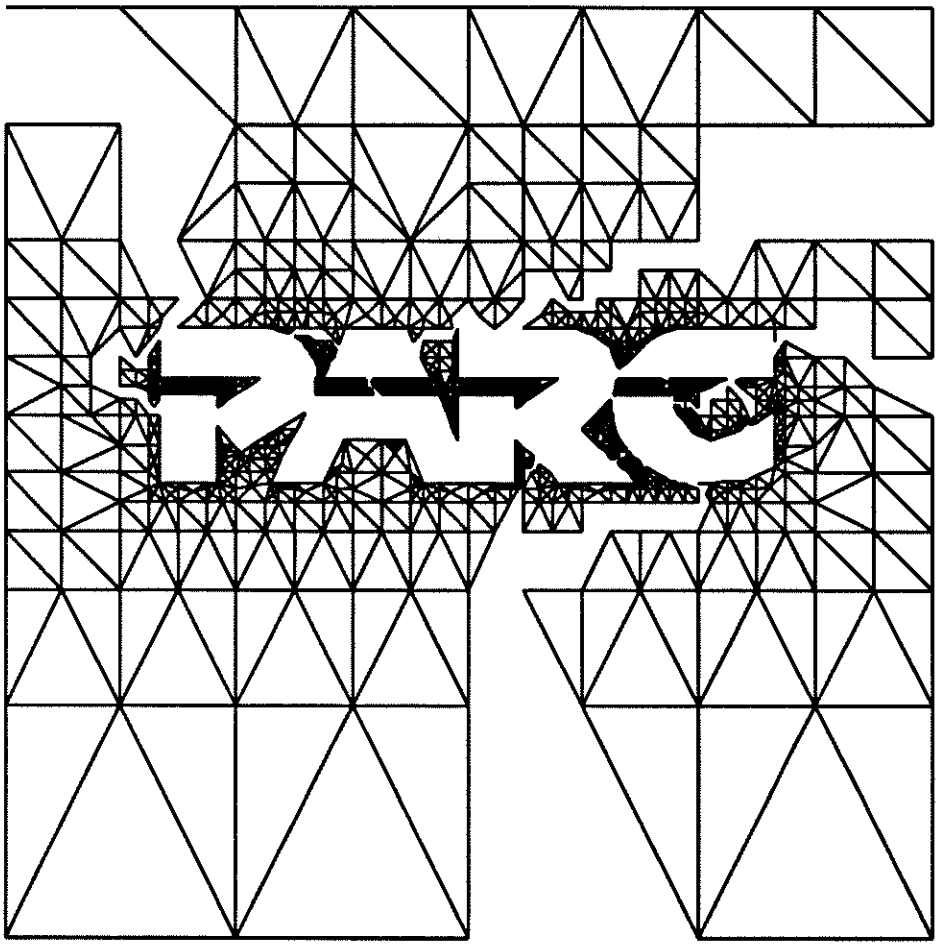

Figure 5: Parc graph partitioned into 12 subsets

p	n_{av}	e_{av} (%)	$d_{av}(G_{ind})$	cc	T_p (s)
4	309	4	2	3	0.02
12	103	6	2	1	0.02
27	46	13	3	2	0.02

Table 3: Parc with 1237 nodes and 3352 edges.

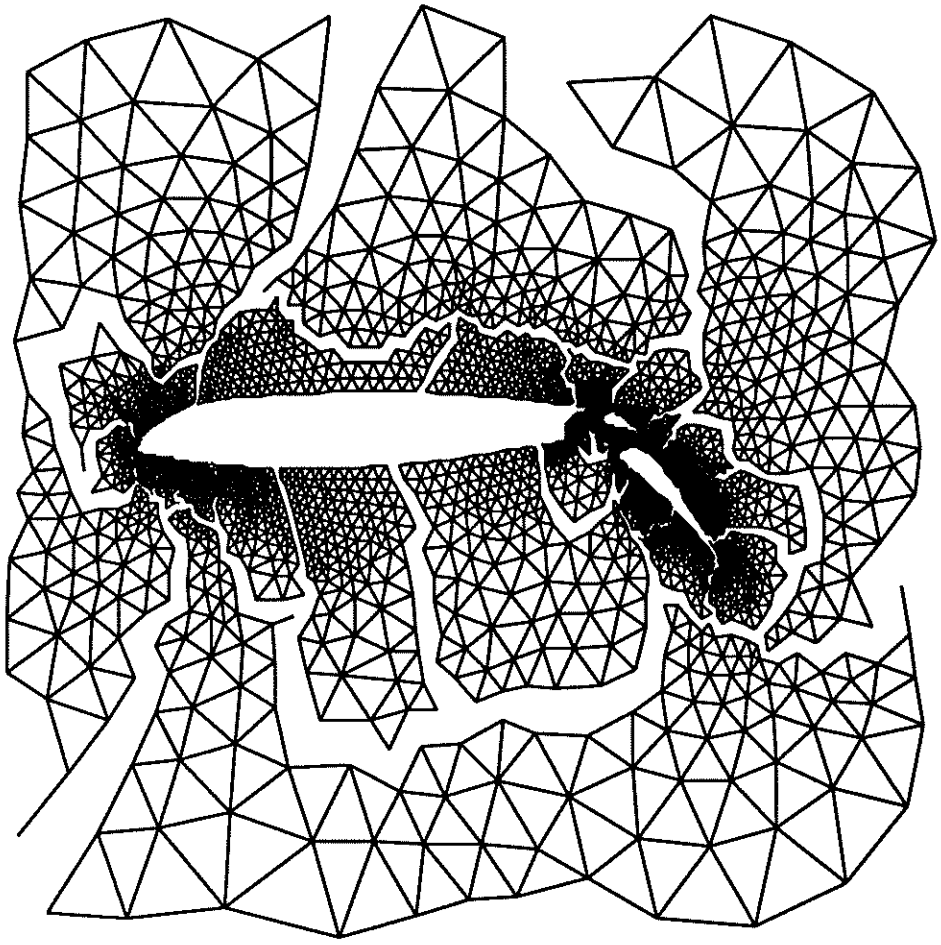

Figure 6: Airfoil graph partitioned into 32 subsets

p	n_{av}	e_{av} (%)	$d_{av}(G_{ind})$	cc	T_p (s)
16	266	7	4	1	0.07
32	133	9	4	1	0.07
128	33	21	5	1	0.08

Table 4: Airfoil with 4253 nodes and 12289 edges.

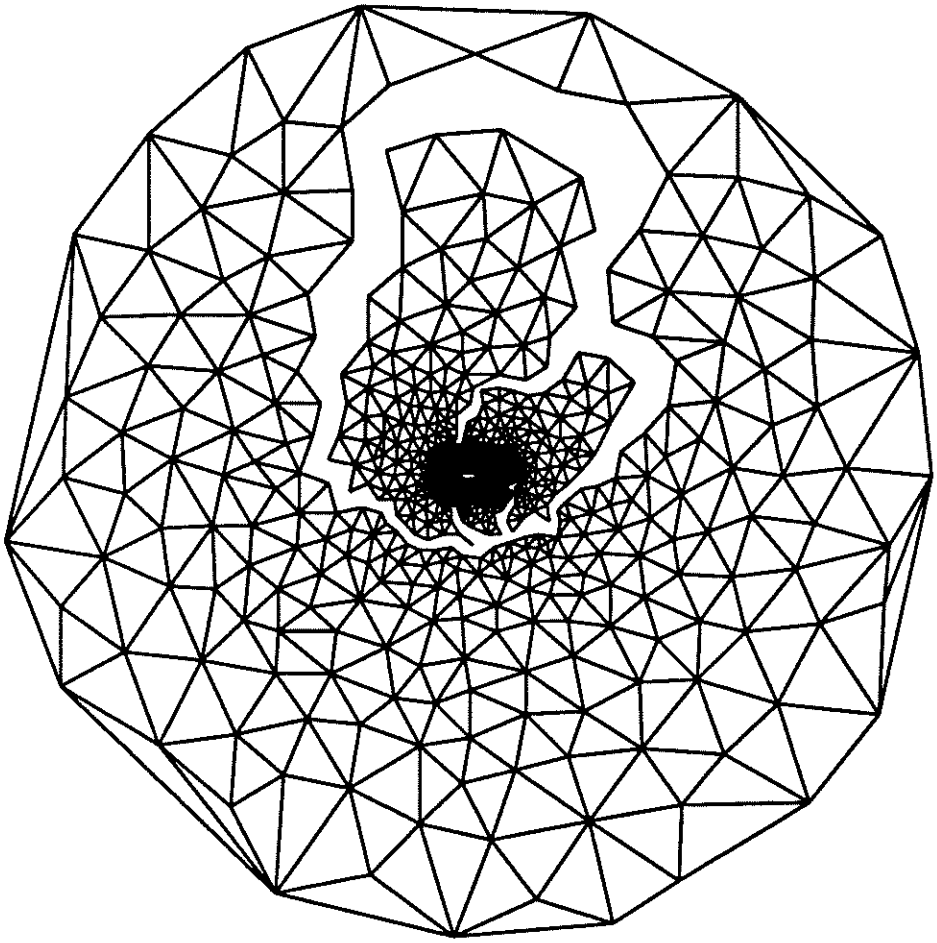

Figure 7: Barth graph partitioned into 30 subsets

p	n_{av}	e_{av} (%)	$d_{av}(G_{ind})$	cc	T_p (s)
30	223	9	4	1	0.13
70	96	14	5	1	0.14
200	33	24	5	2	0.14

Table 5: Barth with 6691 nodes and 19748 edges.

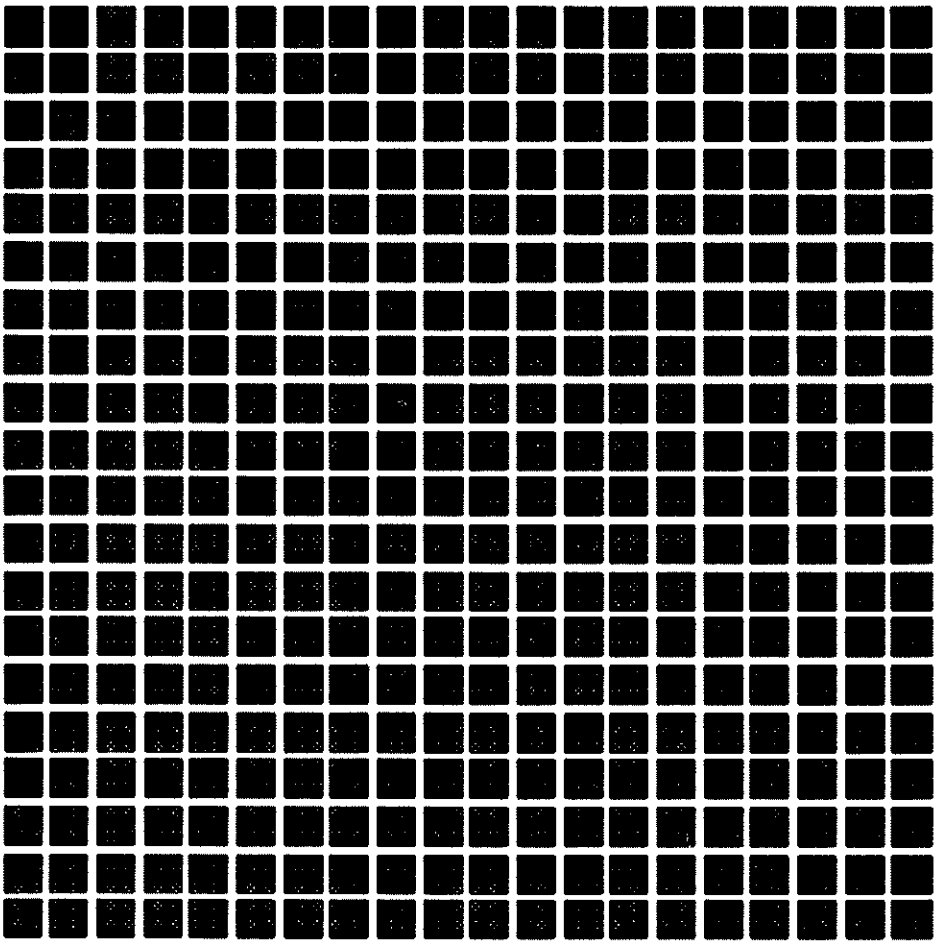

Figure 8: Square graph partitioned into 400 subsets

p	n_{av}	e_{av} (%)	$d_{av}(G_{ind})$	cc	T_p (s)
4	2500	1	2	1	0.24
16	625	4	5	1	0.24
32	313	7	4	1	0.24
50	200	9	5	1	0.25
128	78	16	5	1	0.25

Table 6: Square with 10000 nodes and 39402 edges.

N	M	n_{av}	e_{av} (%)	$d_{av}(G_{ind})$	cc	T_p (s)
4253	12289	8	45	5	1	0.09
16542	48680	32	23	5	2	0.30
65222	193768	127	11	5	2	1.71

Table 7: Airfoil for 512 subsets.

The first five examples are completely irregular graphs. The irregularity does not affect the partitioning algorithm which solves the problem in the order of 1/10 th of a second (Tables 1-5). The percentage of inter-subset edges is not too important. Of course this percentage increases as the ratio N/p decreases. However if we consider significant partitions, i.e. subsets of reasonable sizes (with more than 30 nodes), the percentage of inter-subsets edges represents no more than 25 % of the original communication scheme.

In all these examples the average degree of the induced graph is lower or equal to the average degree of the original graph. This is reassuring and shows in those cases that our partitioning method does not add complexity in terms of the general scheme of communications.

Figures 3 to 7 show a particular partition respectively for the Eppstein graph, the Spiral graph, the Parc graph, the Airfoil graph and the Barth graph.

The last example is given to show that our algorithm, without any trick, produces the result that one could expect. As a matter of fact, the given graph is a discretized square, so the number of nodes is of the form $N = m^2$. Then by partitioning this graph in p subsets where p is like $p = q^2$ and q divides m, we find by using PAM well balanced square subsets, as in Figure 8.

Finally, Table 7 clearly shows that the partitioning time does not depend much, in practice, on the number of nodes and edges. Moreover, for approximatively 200.000 edges the partition is achieved in less than 2 seconds! This shows that our method is reliable independently of the number of edges of a given graph, and that it works very fast even for large graphs.

6 Conclusion

This partitioning method, easy to implement, gives acceptable results at little expense according to the objectives of the GP-problem. We can try to improve it by guessing new starting node or/and tie-break strategy. We can also implement local optimizations to correct results which are too far from the established average value. For example, rebalancing the subsets in terms of the number of nodes or trying to reduce the number of inter-subset edges. More generally, one has to define what the word acceptable means. It certainly depends on the type of problem for which the GP-problem has to be solved. By experiencing different partitioning algorithms, greedy or not, we must be able to put in evidence which algorithm performs better relatively to some priority parameters (percentage of inter-subset edges, time, average degree of the induced graph ...). This will provide the potential user a concrete choice.

However, the knowledge of more theoretical results about the quality of partitioning methods (for example the lower bound of the number of inter-subset edges) would help to define what a good one is. It is certainly in these directions that our future work will go.

References

[1] P. Ciarlet, Jr and F. Lamour. Un algorithme rapide de partitionnement automatique de graphes. Technical report, CEA N-2738, 1993.

[2] C. Farhat. A simple and efficient automatic fem domain decomposer. *Computers and structures*, vol 28, n^o 5, pp 579-602, 1988.

[3] C. Farhat and M. Lesoinne. Automatic partitioning of unstructured meshes for the parallel solution of problems computational mechanics. *International Journal for Numerical Methods in Engineering*, vol 36, n^o 5, pp 745-764, 1993.

[4] B. Hendrickson and R. Leland. An improved spectral graph algorithm for mapping parallel computations. Technical report, SAND 92-1460, 1992.

[5] A. Pothen, H.D. Simon, and K.P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. MATRIX ANAL. APPL.*, vol 11, n^o 3, pp 430-452, 1990.

[6] D.L. Powers. Graph partitioning by eigenvectors. *LINEAR ALGEBRA AND ITS APPLICATIONS*, 101, pp 121-133, 1988.