UCLA
COMPUTATIONAL AND APPLIED MATHEMATICS

On the Validity of a Front-Oriented Approach to Partitioning Large Sparse Graphs with a Connectivity Constraint

P. Ciarlet, Jr.
F. Lamour

December 1994

CAM Report 94-37

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

On the validity of a front-oriented approach to partitioning large sparse graphs with a connectivity constraint

P. Ciarlet, Jr * F. Lamour †

December 12, 1994

Abstract

In this paper we consider the problem of partitioning large sparse graphs, such as finite element meshes. The heuristic which is proposed allows to partition into connected and quasi-balanced subgraphs (see also [8], [9]) in a reasonable amount of time, while attempting to minimize the number of edge cuts. Here the goal is to build partitions for graphs containing large number of nodes or edges, in practice at least 10^4. Basically, the algorithm relies on the iterative construction of connected subgraphs. This construction is achieved by successively exploring clusters of nodes called fronts. Indeed, a judicious use of fronts ensures the connectivity of the subsets at low cost: it is shown that locally, i.e. for a given subgraph, the complexity of such operations grows at most linearly with the number of edges. Moreover, a few examples are given to illustrate the quality and speed of the heuristic.

1 Introduction

Graph partitioning methods are commonly used for a wide variety of problems: problem segmentation, circuit design, parallel computations, domain decomposition methods and many else. Basically, the global problem is decomposed into smaller subproblems which interact as little as possible with one another.

There are different formulations and many variants of the graph partitioning problem, depending on whether one looks at it from the vertex point of view or from the edge point of view. According to the edge point of view, on which we focus in this report, the problem can be expressed mathematically in the following way, for a 2-edge partition or bisection. Let $G = (V, E)$ be a graph where V is the set of nodes and E is the set of edges. The 2-edge partition of G is a partition of V into two disjoint sets V_1 and V_2 such that:

*Commissariat à l'Energie Atomique, CEL-V, D.MA, MCN, 94195 Villeneuve-St-Georges Cedex, France and Department of Mathematics, Univ. of Calif. at Los Angeles, CA 90024. E-mail: ciarlet@limeil.cea.fr, ciarlet@math.ucla.edu. The work of this author was partially supported by the DGA/DRET under contract 93-1192 and by the Army Research Office under contract DAAL03-91-C-0047 (Univ. Tenn. subcontract ORA4466.04 Amendment 1).

†Cisi, Branche CEA et Défense, BP 28, 91192 Gif-sur-Yvette Cedex, France and Department of Mathematics, Univ. of Calif. at Los Angeles, CA 90024. E-mail: lamour@limeil.cea.fr, lamour@math.ucla.edu. The work of this author was partially supported by the National Science Foundation under contract ASC 92-01266, the Army Research Office under contract DAAL03-91-C-0047 (Univ. Tenn. subcontract ORA4466.04 Amendment 1), and ONR under contract ONR-N00014-92-J-1890.

- $V = V_1 \bigcup V_2$,
- $\max(|\ V_1\ |, |\ V_2\ |) \leq \alpha\ |\ V\ |$,
- $|E_0|$ is as small as possible, where $E_0 = E \cap V_1 \times V_2$ is the set of edge cuts or intersubset edges.

Here, we use the classical notation $|.|$ for the cardinality of a set. The value of the constant α ($\frac{1}{2} \leq \alpha < 1$) characterizes the looseness in the balancing of the partition. It is well known that the previous problem is NP–complete for any value of the parameter α. Thus, many heuristics have been designed to approximate as closely as possible this bisection problem. Among them, nodes interchanging methods [20], [13], branch and bound methods [7], [24], maximum flows [6], maximum matching [21] or eigenvalue computations [23], [16].

In the general case, the p-edge partition of G, for $p \geq 2$, is a partition of V into p disjoint subsets $V_1, V_2, \cdots, V_p$ such that:

- $V = \bigcup_{i=1}^{p} V_i$,
- $|V_i| \approx |V_j|$,
- $|E_0|$ is as small as possible, where $E_0 = E \cap \bigcup_{i \neq j} V_i \times V_j$.

This more general problem is also NP–complete, and many heuristics have likewise been constructed to approximate it. First, there is an important class of such methods which rely on one of the bisection heuristics of the previous paragraph. Initially, the graph is partitioned into two parts and then each subgraph is divided into two, and the process goes on recursively. Among them, eigenvalue computations [2], [17] or multilevel techniques [19]. Many other heuristics have been designed to deal directly with the general p-edge partition problem, such as linear programming transportation [3], greedy algorithms [12], [10], nodes interchanging methods [4] or geometrical approaches [22], [1].

Note that some of these methods can be utilized jointly. Particularly, nodes interchanging methods require some initial partition. On the other hand, these methods are often used as a postprocessing step, i.e. after a first approximation to the solution to the p-edge partition problem has been determined, because they generally increase the quality of the partition.

In the following, we complement the greedy method introduced in [9] by a new load-balancing step and give a general framework to these heuristics. Both methods are based on particular clusters of nodes which we call *fronts*. The next Section recalls some graph definitions and notations together with the p-edge partitioning and load-balancing problems. Section 3 is devoted to fronts and connectivity in graphs. In Sections 4 and 5, the partitioning heuristic and the post processing steps are described thoroughly. In the next Section, some experiments are listed on a number of graph examples. Finally, we summarize the main results in the Conclusion.

2 Notations and problems

First, let us recall a few definitions about graphs and set the notations that will be used throughout this paper. For more details, we refer the reader to [5]. Let $G = (V, E)$ be an *undirected* graph where V is the set of nodes and E the set of edges. Only *simple* graphs will be considered hereafter, i.e. without multiple edges or loops. Let $N = |V|$ and $M = |E|$ be the number of nodes and edges respectively.

G is said to be *connected* if, for any pair of nodes (v, w), either (v, w) is an edge or there exists k nodes (x_i) such that the edges of the *path* (v, x_1), (x_1, x_2), ..., (x_{k-1}, x_k), (x_k, w) belong to E. Otherwise, G is said to be *multiconnected*.

For a node v, let $\Gamma(v)$ denote the set of its *neighbors*, that is the set of nodes w such that (v, w) belongs to E, and let $\underline{\Gamma}(v) = \Gamma(v) \cup \{v\}$. More generally, for a nonempty subset W of V, $\underline{\Gamma}(W) = \cup_{w \in W} \underline{\Gamma}(w)$ and, by extension, $\underline{\Gamma}^l(W) = \underline{\Gamma}(\underline{\Gamma}^{l-1}(W))$ for $l \geq 2$. For practical reasons, we

define $\underline{\Gamma}^0(W) = W$. The *degree* of a node v, denoted by $d(v)$, is $|\Gamma(v)|$. For a node v and a subset W, let $d_W(v)$ be the number of neighbors of v belonging to W. Finally, let $\bar{d}$ be the average degree of a node in G. By definition, $\bar{d} = 2M/N$.

For any pair of nodes (v, w), the *distance* $\delta_G(v, w)$ corresponds to the length (i.e. number of edges) of the shortest path linking v to w in G ($\delta_G(v, v) = 0$); by extension, if there is no path linking two nodes v and w in the graph, then $\delta_G(v, w) = +\infty$ [5]. For a nonempty subset W and a node v, let $\delta_G(W, v) = \min_{w \in W} \delta_G(w, v)$. Finally, $\max_{v,w} \delta_G(v, w)$ is called the *diameter* of the graph.

In the following, as we partition graphs that come from physical meshes, usually two- or three-dimensional, we talk about the *boundary* of the graph. Let n_b be the number of boundary nodes.

For the partitioning problem considered here, we shall denote by p the number of subsets V_i and $n_i = |V_i|$. Let $\bar{n}$ be the average of (n_i), that is $\bar{n} = N/p$. Moreover, for $1 \leq i \leq p$, let $E_i = E \cap V_i \times V_i$ and $G_i = (V_i, E_i)$ and let $E_0 = E \cap \bigcup_{i \neq j} V_i \times V_j$. Note that $E_0, E_1, \cdots E_p$ is a disjoint partition of E. If we let $m_i = |E_i|$ and $m_0 = |E_0|$, then the p-edge partitioning problem we solve hereafter is:

Find p *disjoint* subsets $V_1, V_2, \cdots, V_p$ of V such that

- $V = \bigcup_{i=1}^{p} V_i$,
- $n_i \approx n_j$,
- the subgraphs G_i are *connected*,
- m_0 is minimized.

Remark 1 *Note that a connectivity constraint has been added to the usual p-edge partitioning problem. This constraint is well suited to the numerical problems generally solved from the partition of a finite element mesh.*

Because of the added connectivity constraint, it often happens that the subsets that are obtained, although connected, are not balanced. Thus a postprocessing heuristic, based on reassigning nodes or sets of nodes is used afterwards. Two load balancing steps, called LB1 and LB2, aim at decreasing the value of the standard deviation of (n_i), i.e. $\sigma = \sqrt{\frac{1}{p}\sum_{i=1}^{p}(n_i - \bar{n})^2}$. LB1, presented in [10], relies on successive *single* node reassignments, enforcing the connectivity constraint on the subgraphs and trying to minimize m_0. LB2, on the other hand, allows for successive *multiple* nodes reassignments (fronts), enforcing only the connectivity constraint.

3 Fronts in graphs

There are two obvious ways of exploring a graph G, from a given vertex v:

- examine all its neighbors and then pass to one of them, or
- examine one of its neighbors w and then pass to w.

The first approach is called *breadth-first search* and the second *depth-first search* (see [15] for a thorough description). In this paper, we consider only algorithms based on breadth-first-like search. The idea is to explore the graph by *fronts*. From a nonempty subset W of V (particularly singletons), examine all nodes at distance 1, then all nodes at distance 2, etc. Fronts at distance k, $k \geq 0$, of W are defined by

$$F(W, k) = \{v \in V,\ \delta_G(W, v) = k\}.$$

Proposition 1 *For a nonempty subset W of V and $k \geq 1$, $\underline{\Gamma}^k(W) = \underline{\Gamma}^{k-1}(W) \cup F(W,k)$ and $\underline{\Gamma}^{k-1}(W) \cap F(W,k) = \emptyset$.*

The proof proceeds by induction.
For $k = 1$, by definition,

$$\begin{aligned} \underline{\Gamma}(W) &= \cup_{w \in W} \underline{\Gamma}(w) = \cup_{w \in W} \Gamma(w) \cup W \\ &= \{v \in V,\ \exists w \in W,\ (v,w) \in E\} \cup W \\ &= \{v \in V \setminus W,\ \exists w \in W,\ (v,w) \in E\} \cup W \\ &= \{v \in V,\ \delta_G(W,v) = 1\} \cup W \\ &= F(W,1) \cup \underline{\Gamma}^0(W). \end{aligned} \tag{1}$$

Moreover, $\underline{\Gamma}^0(W) = W$ and $F(W,1) = \{v \in V \setminus W,\ \delta_G(W,v) = 1\}$. Thus there are disjoint.

Now, let us assume that the result is true for $l \leq k-1$ and prove it also holds for k. First, by assumption, we have

$$\begin{aligned} \underline{\Gamma}^{k-1}(W) &= \underline{\Gamma}^{k-2}(W) \cup F(W,k-1) = \cdots = \cup_{l=0}^{k-1} F(W,l) \\ &= \{v \in V,\ \delta_G(W,v) \leq k-1\} \end{aligned} \tag{2}$$

Thus $\underline{\Gamma}^{k-1}(W) \cap F(W,k) = \emptyset$. If we use (1) and replace W by $\underline{\Gamma}^{k-1}(W)$, then we have

$$\begin{aligned} \underline{\Gamma}(\underline{\Gamma}^{k-1}(W)) &= F(\underline{\Gamma}^{k-1}(W),1) \cup \underline{\Gamma}^0(\underline{\Gamma}^{k-1}(W)), \text{ or} \\ \underline{\Gamma}^k(W) &= F(\underline{\Gamma}^{k-1}(W),1) \cup \underline{\Gamma}^{k-1}(W). \end{aligned}$$

To complete the proof, one has to check that $F(\underline{\Gamma}^{k-1}(W),1) = F(W,k)$:

$$\begin{aligned} F(\underline{\Gamma}^{k-1}(W),1) &= \{v \in V \setminus \underline{\Gamma}^{k-1}(W),\ \exists z \in \underline{\Gamma}^{k-1}(W),\ \delta_G(z,v) = 1\} \\ &= \{v \in V,\ \delta_G(W,v) > k-1,\ \exists z,\ \delta_G(W,z) \leq k-1,\ \delta_G(z,v) = 1\} \text{ by (2)} \\ &= \{v \in V,\ \delta_G(W,v) = k\}. \bullet \end{aligned}$$

By definition, once a node has been reached and processed, it is *marked*. Then, by rewriting the Proposition in the following way, $F(W,k) = \underline{\Gamma}^k(W) \setminus \underline{\Gamma}^{k-1}(W)$, one gets a natural way of traveling simply through the graph by successively exploring fronts, where each new front is the set of neighbors which have not been marked yet. As for breadth-first search (and depth-first search), the complexity of exploring the graph by fronts is $O(M)$, as each edge is considered exactly twice during the process.

We shall use this method by fronts in the next Section to build a partitioning heuristic. Now, as our goal is to obtain connected subsets of the set of nodes, let us prove a few simple yet powerful results regarding the connectivity: more precisely how to check that a graph or a subgraph is connected with the help of fronts.

Proposition 2 *A graph $G = (V,E)$ is connected if and only if either*

1. *There exists a connected subset W_c of V such that the expansion by fronts from W_c spans V, or*
2. *For all subsets W of V, the expansion by fronts from W spans V.*

(i) if G is connected, then all distances are finite and 2 is true.
(ii) if 2 is satisfied, then 1 is clearly verified.

(iii) if 1 holds, then let us check that G is connected. Let (v_1, v_2) be a pair of nodes. As the expansion by fronts from W_c spans V, there exist two nodes (w_1, w_2) belonging to W_c and two paths in G linking v_1 to w_1 and v_2 to w_2. By assumption, as W_c is connected, there exists a path linking w_1 to w_2. Thus there is a path in G linking v_1 to v_2: G is connected. •

Remark 2 *When W_c is connected, then for all positive k, $F(W_c, k)$ is connected by construction.*

Now, let $G = (V, E)$ be a connected graph and W a subset of V. Let

- $V_- = V \setminus W$, $E_- = E \cap V_- \times V_-$, $G_- = (V_-, E_-)$,
- $G_\Gamma = (\Gamma(W), E_- \cap \Gamma(W) \times \Gamma(W))$ and
- s a vertex of $\Gamma(W)$.

In the next proposition, we give a few conditions to check whether G_- is connected or not.

Proposition 3 *G_- is connected if either*

1. *The expansion by fronts from s in G_- spans V_-, or*
2. *The expansion by fronts from s in G_- spans $\Gamma(W)$, or*
3. *The expansion by fronts from s in G_Γ spans $\Gamma(W)$.*

Moreover, the connectivity of G_- implies conditions 1 and 2 but does not imply condition 3.

(i) Clearly, the fact that G_- is connected implies condition 1. Also, condition 1 implies condition 2.

(ii) Let us prove that condition 2 yields the connectivity of G_-. Let (v, w) belong to $V_- \times V_-$. G is connected by assumption and therefore there exists a path $(x_1, \cdots, x_k)$ of nodes of V such that (v, x_1), ..., (x_k, w) are in E. If no x_i belongs to W, then the path is in G_-. Now, if there is at least one x_i in W, let i_0 (resp. i_1) denote the first (resp. last) index such that $x_i \in W$. If we let $x_0 = v$ and $x_{k+1} = w$, then x_{i_0-1} and x_{i_1+1} belong to $\Gamma(W)$ and by assumption (condition 2), there exists a path $y_1, \cdots, y_l$ in G_- linking x_{i_0-1} to x_{i_1+1}. Then the path $x_1, \cdots, x_{i_0-1}, y_1, \cdots, y_l, x_{i_1+1}, \cdots, x_k$ is in G_- and links v to w: G_- is connected.

(iii) G_Γ is embedded in G_-, thus condition 3 implies condition 2.

(iv) Finally, to see that condition 2 does not yield condition 3, one can consider the example of the removal of a single node from an annulus (i.e. a node is exactly one endpoint of two edges) of size at least four. •

The complexities $(C_i)_{i=1,2,3}$ of checking the connectivity of G_- by properties 1 to 3 of Proposition 3 are $C_1 = C_2 = O(M_-)$ and $C_3 = O(M_\Gamma)$, and in any case $C_3 \leq C_2 \leq C_1$.

Remark 3 *To prove these two Propositions, we mainly used the equivalence between the connectivity of a graph and the finiteness of distances in it.*

4 Greedy Partitioning heuristics

Briefly there are two schools for computing a solution of the p-edge partitioning problem for large graphs: from the matrix point of view, i.e. using the Laplacian matrix [14] associated to the graph, or directly from the graph itself. The first point of view is based on spectral properties of the Laplacian matrix. Among the different works, let us mention those of Barnes [3], Simon [25] and Hendrickson and Leland [19]. The other way is to work directly on the graph by applying greedy methods: the

heuristic presented below is based on such a greedy algorithm. Let us mention that the first greedy algorithm for partitioning finite element meshes was proposed by Farhat [12].

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 p-edge partitioning problem. In this case, a greedy algorithm can be described as an algorithm that computes one after the other each subset V_i by simply accumulating nodes when traveling through the graph. Evidently, we choose to travel in the graph by fronts. Then the only problematical questions are: how to start and how to stop?

A starting node v_s is chosen and marked. The accretion process is done by successively building fronts from v_s and so on as long as the expected total number of nodes is not reached.

The way of selecting 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 selects the prescribed number of nodes among all the candidate nodes of the last front contributes to the quality of the final partition.

Thus, for each subset, one can summarize the greedy procedure by

1. Select 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 p-edge partitioning heuristics. However, if we suppose that the tie-break strategy only involves *sorting*, then an obvious justification of using greedy heuristics is that they are inexpensive, as we can see in the following Proposition.

Proposition 4 *The overall complexity of a greedy algorithm applied to the p-edge partitioning problem is* $O(M) + \{\max(p, \log_2(\frac{N}{p}))\}O(N)$.

By using the above procedure, one can easily see that the complexities of steps 1, 2 and 3 are respectively $O(p.N)$, $O(M)$ and $O(p.\bar{n}\log(\bar{n}))$. This proves the result. •

Remark 4 *The optimal choice for the tie-break strategy is: select the required number of nodes among the candidates of the last front in such a way that the number of edge cuts is minimized, i.e. the number of edges between the current subset and the rest of the graph. It is clear that this optimal choice is NP–complete. Thus the need for an approximate choice based on sorting to reduce the complexity.*

4.1 The GP heuristic

First, let us recall that, as we consider meshes, we can use the notion of the boundary of the graph.

As greedy algorithms do, our algorithm builds iteratively the different subsets of the partition by accumulating nodes in each subset and marking them when they have been selected. For the first iteration, the starting node is chosen among the boundary nodes with minimum degree. On the other hand, 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, the number of boundary nodes. In that case the new current boundary is the set of unmarked nodes that are the

neighbors of marked nodes. From there each new starting node is chosen among nodes which stand on the current boundary and are neighbors of the previously built subset. Moreover this node is preferably chosen among the candidates with minimum *current degree*, i.e. number of unmarked neighbors. Note that the current degree starts from the degree of the node and decreases to zero as the partitioning algorithm goes along.

A way to redefine the current boundary at low cost is to introduce the notion of *virtual boundary*. Actually, this virtual boundary is defined, after 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.

We consider the case of a connected graph G and prove results concerning its properties and its complexity. The greedy partitioning algorithm "GP" is described next.

Let $i = 1$.

1. If $i < p$
   (a) Compute $n_i = \frac{N - \sum_{j=1}^{i-1} n_j}{p-(i-1)}$.
   (b) Select an unmarked node v_s such that:
      i. v_s belongs to the current boundary,
      ii. if the current boundary is not new, v_s is a neighbor of a node belonging to V_{i-1} (if possible[1]),
      iii. v_s has a minimal current degree.

      Mark v_s and initialise V_i with v_s.
   (c) $K = 0$.

      If there are unmarked neighbors of nodes of V_i then
      i. let K be their number.
      ii. If $\mid V_i \mid + K < n_i$ then
         mark those nodes and add them to V_i,
         update the current degree of their neighbors,
         return to 1.(c).
   (d) If $\mid V_i \mid + K \geq n_i$ then
      mark $(n_i - \mid V_i \mid)$ minimal current degree nodes and add them to V_i,
      update the current degree of their neighbors,
      update the current and virtual boundaries,
      do $i = i + 1$ and return to 1.
   (e) If there are no more unmarked neighbors of nodes of V_i then
      unmark the nodes in V_i and assign them to neighboring subsets,
      return to 1.
2. Mark all the remaining nodes and add them into V_p. If V_p is multiconnected then keep the largest component and unmark the nodes of the other components and assign these nodes to neighboring subsets.

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

Remark 5 *This heuristic can be used for any graph, that is without the notion of boundary, be it current or virtual. Nevertheless, numerous numerical experiments show that, when compared to the original heuristic for physical meshes, it works slightly slower, and gives worse results in terms of edge cuts.*

4.2 Tie-break strategy

Let us go into the details of our tie-break strategy when building a given subset V_{i_0}. Note that because of its intrinsic complexity, the optimal choice is impossible to determine. Suppose that the k^{th} front $F(v_s, k)$ contains too many nodes $(v_i)_{i=1,\cdots p_k}$ and that we want only q of them. The optimal choice corresponds to selecting q nodes such that the number of edges between them and nodes that do not belong to the current subset (i.e. $d(v_i) - d_{V_{i_0}}(v_i)$ for one node) is minimized. Then the objective is to find nodes q nodes $(v_{\sigma(j)})$ minimizing

$$f_{opt}(v_{\sigma(j)}) = \sum_{j=1}^{q} \{d(v_{\sigma(j)}) - d_{V_{i_0}}(v_{\sigma(j)})\}.$$

If we let $d_c^l(v)$ be the current degree of a node v after the l^{th} front is obtained, then $d_c^k(v) = d(v) - \sum_{i \leq i_0} d_{V_i}(v)$. Thus the objective can be reformulated as

$$f_{opt}(v_{\sigma(j)}) = \sum_{j=1}^{q} \{d_c^k(v_{\sigma(j)}) + \sum_{i<i_0} d_{V_i}(v_{\sigma(j)})\}. \tag{3}$$

Here, we make two modifications to get a simpler objective function. First, we can neglect the contribution coming from other subsets, as this contribution appears anyway in the total number of edge cuts. Indeed, a node v which has not been assigned yet necessarily belongs to V_i, for $i \geq i_0$. Then the edges corresponding to $\{\sum_{i<i_0} d_{V_i}(v)\}$ will be cut. In other words, it is possible to suppress the second term from (3). By doing this, the complexity of the tie-break strategy has not been modified. Therefore, in the second place, we have to approximate the remaining term: the terms $d_c^k(v_{\sigma(j)})$ are actually determined only after all the nodes have been chosen. On the other hand, the quantities $d_c^{k-1}(v_{\sigma(j)})$ are already at hand. Then there are two obvious ways of approximating (3). The first one is to replace it by

$$f(v_{\sigma(j)}) = \sum_{j=1}^{q} d_c^{k-1}(v_{\sigma(j)}), \tag{4}$$

which amounts to sorting the nodes of the last front by increasing current degree (before they are taken into account) and to keeping the first q. Thus this process can be rewritten as:

Sort the nodes by increasing actualized degree: $(v_{\sigma(j)})_{j=1,\cdots,p_k}$.
Keep $(v_{\sigma(j)})_{j=1,\cdots,q}$. (TB1)

The second way is to choose one node after the other: a node with the lowest current degree is selected, then the current degree of the remaining nodes is updated, a second node is chosen and the process goes on. Thus it can be summarized as:

Sort the nodes by increasing actualized degree: $(v_{\sigma(j)})_{j=1,\cdots,p_k}$ and keep $v_{\sigma(1)}$.
For $l = 2$ to q
    Update the current degree of $(v_{\sigma(j)})_{j=l,\cdots,p_k}$ and keep $v_{\sigma(l)}$. (TB2)

It is clear that, in terms of edge cuts, TB1 produces more cuts than TB2 and that, in terms of complexity, TB2 is more expensive than TB1. As stated in the algorithm describing GP, we have kept the tie-break strategy TB1.

By construction, the heuristic proposed in this paper relies mainly on the iterative construction of the subsets via the fronts, and less on the tie-break strategy. That is the reason why it should be applied mainly to *sparse* graphs, because then it is possible to get multiple fronts before having to break the tie when enough nodes have been aggregated. On the contrary, for a graph with *high* average degree (or equivalently a *small* diameter), it happens that the tie-break strategy has to be used right from the start, i.e. without building even a single front but instead having to select some of the nodes of the first front.

4.3 Finiteness, effectiveness and complexity

Let us show that, when GP is applied to the p-edge partitioning problem for a connected graph G, it indeed builds p connected subsets.

Theorem 1 *The algorithm GP terminates and always builds p connected subsets.*

First of all, let us prove that the algorithm terminates. Note that for this, we simply have to verify that the value of i is incremented in a finite number of operations (Loop 1). Step 1.(a) is straightforward. For step 1.(b), one has to verify that a node v_s can be always be selected. For this, it suffices to check that the current boundary can not be empty: this stems from its definition. If $i = 1$, then it is the original boundary. On the other hand, for $i > 1$, it is clear that there is at least one unmarked node neighboring a marked one: it belongs to the current boundary which is therefore not empty. Finally, one goes through steps 1.(c), 1.(d) and 1.(e) in a finite number of operations (using front techniques and sorting).

In order to prove that it builds exactly p connected subsets, let us proceed by contradiction. Suppose that $\tilde{p}$ connected subsets are constructed, $\tilde{p} \neq p$, when the algorithm terminates. Due to the structure of GP (iterate while $i < p$ and finally build V_p), it is clear that $\tilde{p} < p$. This implies that there exists i, $1 \leq i \leq p$, such that $n_i = 0$.
If $i = p$, then let us prove that we also have $n_{p-1} = 0$ (here, we want to make use of the body of Loop 1 and therefore need to prove that $i \leq p - 1$).
As $n_p = 0$ and $V = \bigcup_{i=1}^{n} V_i$,

$$N = \sum_{j=1}^{p-1} n_j. \tag{5}$$

Now, using step 1.(a) of the algorithm for $i = p-1$ gives that $n_{p-1} = (N - \sum_{j=1}^{p-2} n_j)/2$, or equivalently

$$2n_{p-1} \leq (N - \sum_{j=1}^{p-2} n_j). \tag{6}$$

This with (5) yields $n_{p-1} = 0$. Thus there exists i, $1 \leq i \leq p-1$, such that $n_i = 0$. Without changing notations, suppose that i is the smallest index such that $n_i = 0$, i.e. $n_j > 0$ for $1 \leq j \leq i-1$. Note

that V_1 can always be built because G is connected; so $i > 1$.
Then, when $n_i = 0$ is computed ($i < p$), it comes after either step 1.(d) for $i-1$ or step 1.(e) for i.
Next we shall prove that a contradiction can be reached for these two cases.
In the first case, both the definition of n_{i-1} and the inequality $n_{i-1} \geq 1$ (by assumption on index i) imply

$$[p-(i-2)] \leq n_{i-1}.[p-(i-2)] \leq N - \sum_{j=1}^{i-2} n_j.$$

Therefore,

$$\begin{aligned}
[N - \sum_{j=1}^{i-1} n_j].[p-(i-2)] &= [N - \sum_{j=1}^{i-2} n_j].[p-(i-2)] - n_{i-1}.[p-(i-2)] \\
&\geq [N - \sum_{j=1}^{i-2} n_j].[p-(i-1)] \\
&\geq [p-(i-2)].[p-(i-1)], \text{ or} \\
N - \sum_{j=1}^{i-1} n_j &\geq [p-(i-1)]. \qquad (7)
\end{aligned}$$

Step 1.(a) and (7) yield $n_i \geq 1$, a contradiction.

In the second case, we denote by n_j^{prev} the quantities obtained before step 1.(e), i.e. before reassigning the nodes of V_i, and n_j^{curr} the current ones. Let us note that necessarily $n_i^{prev} \geq 2$, because otherwise step 1.(e) cannot be reached (if $n_i^{prev} \leq 1$, the last step was 1.(d)). Thus

$$2[p-(i-1)] \leq n_i^{prev}.[p-(i-1)] \leq N - \sum_{j=1}^{i-1} n_j^{prev}.$$

Finally, denote by n_i^{conn} the number of reassigned nodes. Then

$$n_i^{conn} < n_i^{prev} \quad \text{and} \quad \sum_{j=1}^{i-1} n_j^{prev} + n_i^{conn} = \sum_{j=1}^{i-1} n_j^{curr}.$$

Therefore

$$\begin{aligned}
[N - \sum_{j=1}^{i-1} n_j^{curr}].[p-(i-1)] &= [N - \sum_{j=1}^{i-1} n_j^{prev} - n_i^{conn}].[p-(i-1)] \\
&\geq [N - \sum_{j=1}^{i-1} n_j^{prev} - n_i^{prev}].[p-(i-1)] \\
&\geq [N - \sum_{j=1}^{i-1} n_j^{prev}].[p-i] \\
&\geq 2[p-(i-1)].[p-i], \text{ implying} \\
N - \sum_{j=1}^{i-1} n_j^{curr} &\geq [p-(i-1)]. \qquad (8)
\end{aligned}$$

Step 1.(a) and (8) yield $n_i^{curr} \geq 1$, again a contradiction. •

Let us prove now that GP is inexpensive.

Theorem 2 *The complexity of GP is $O(M)$.*

In order to obtain this result, let us consider the complexity of the algorithm step by step for the construction of a given subset V_i, $i < p$.

The complexity of step 1.(a) is $O(1)$.

For step 1.(b), we shall consider two cases, depending on whether the current boundary is new or not.

If the j^{th} current boundary is new, then v_s is chosen in $O(n_b^j)$ operations, using 1.(b).i. and 1.(b).iii. Note that a node belongs to at most one current boundary, so $\sum_j n_b^j \leq N$. Moreover, if the current boundary is multiconnected, then the choice is made in $O(n_b^{j,l})$ operations for the l^{th} connected component, with $\sum_l n_b^{j,l} = n_b^j$.

If the current boundary is not new, then let us divide the operations count into two subcases, according to whether it comes after step 1.(d) for $i-1$ or step 1.(e) for i.
In the first subcase, it is a simple matter to check that v_s can be chosen in $O(\sum_{v \in V_{i-1}} d_c(v))$ operations, by using 1.(b).ii and then 1.(b).i and 1.(b).iii.
If the second subcase happens, then necessarily it comes after a first subcase for which we suppose that a sorted list of candidates has been built [2]. Thus, to find v_s, it suffices to update this sorted list, which can be done in $O(\sum_{v \in V_i^{prev}} d_c(v))$ operations.

By dividing the case in this way, note that a given node appears at most twice in the summation (once while examining its unmarked neighbors, and once building the sorted list).

For step 1.(c), it is clear that it is done in $O(m_i)$ operations (construction by fronts).

In step 1.(d), updating the current and virtual boundaries is done respectively in $O(n_i)$ and $O(m_i)$ operations and the tie-break strategy requires $O(n_i)$, if one again takes the sorting of the footnote into account.

Finally, if the occasion should arise (step 1.(e)), nodes are reassigned in $O(m_i^{prev})$ operations by successively reassigning each of them (in reverse order) to the subset which holds the highest number of its neighbors.

Clearly, the complexity of step 2. is $O(m_p)$.

To recapitulate, the complexities of the p steps 1.(a), 1.(b), 1.(c), 1.(d), 1.(e) and of step 2. are respectively bounded by:

$$\begin{array}{ll} O(p), & O(\sum_j n_b^j + O(\sum_i \sum_{v \in V_i} d_c(v)), \\ O(\sum_i m_i), & O(\sum_i n_i + \sum_i m_i), \\ O(\sum_i m_i), & O(m_p). \end{array}$$

The conclusion follows if one uses $O(\sum_i \sum_{v \in V_i} d_c(v)) \leq O(\sum_i m_i) \leq O(M)$. •

5 Postprocessing steps

As previously mentioned, the added connectivity constraint often produces not balanced subsets, in terms of (n_i). Thus the need for postprocessings whose goal is to balance the subsets, i.e. decrease the

[2] this can be done in $O(\sum_{v \in V_{i-1}} d_c(v))$ operations, if one recalls that the nodes are sorted by current degree. Indeed, the current degree of a node varies between 0 and d_{max}, therefore sorting n of them is done in $O(n)$ operations by piling them up in $(d_{max}+1)$ heaps.

value of $\sigma = \sqrt{\frac{1}{p}\sum_{i=1}^{p}(n_i - \bar{n})^2}$. In this respect, it differs from more classical post-treatments which come after well balanced subsets have been obtained: in that case, the emphasis is on reducing the number of edge cuts m_0. Hereafter, we briefly recall some of these classical methods and then consider our balancing heuristics. All these postprocessing steps are based on node interchanging.

5.1 Reducing the number of edge-cuts

To our knowledge, the first heuristic was proposed by Kernighan and Lin [20]. In that case, the quality of a *balanced* 2-edge partition $V_1 \cup V_2$ is improved by exchanging subsets of V_1 and V_2. The selection criterion of the subsets is determined from a gain function which is defined as follows. First, for $v \in V_1$, $g_v = d_{V_2}(v) - d_{V_1}(v)$ and for $w \in V_2$, $g_w = d_{V_1}(w) - d_{V_2}(w)$. Then the gain obtained by exchanging a node $v \in V_1$ with a node $w \in V_2$ is equal to:

$$g_{v,w} = g_v + g_w - 2\delta(v,w)$$

where $\delta(v,w)$ is defined by:

$$\delta(v,w) = \begin{cases} 1 & if\ (v,w) \in E \\ 0 & otherwise \end{cases}.$$

So, after computing for each node $v \in V_1$ and $w \in V_2$ the values of g_v and g_w, the algorithm chooses a pair of nodes (v_1, w_1) which maximizes the gain g_{v_1,w_1}. Then, for all neighbors u of v_1 or w_1 the values of g_u are updated. The optimizing process is iterated until $g_{v_{n-1},w_{n-1}}$ is computed. Finally, the algorithm chooses among all pairs of subsets $\{v_1, v_2, ..., v_k\}$, $\{w_1, w_2, ..., w_k\}$ for $k \in \{1, ..., n-1\}$ the one whose swapping corresponds to the maximum of the gain $\sum_{l=1}^{k} g_{v_l,w_l}$.

This process can be repeated iteratively until no better improvement, regarding to the number of intersubset edges, is obtained. The complexity of one iteration is $O(N^2 \log N)$. The number of iterations is not bounded *a priori*.

Many variants of this heuristic have been proposed in the past years which aim at reducing the overall compexity. Fiduccia and Mattheyses [13] remarked that the high cost of Kernighan and Lin's heuristic is due to the $\delta(v,w)$ term in the expression of $g_{v,w}$. Thus, instead of selecting nodes by pairs, they proposed to choose them one after the other, that is v_1, then w_1, then v_2 etc. Then the gain is simply :

$$g'_{v,w} = g_v + g_w$$

where g_w takes into account the fact that v has already been moved from V_1 to V_2. Again, the process can be repeated. The complexity of one iteration is reduced to $O(M)$. Still, the number of iterations is not bounded.

Recently, Hendrickson and Leland, [18] and [19], generalized Fiduccia and Mattheyses' method to an arbitrary number of subsets. Instead of a single gain function there are $(p-1)$ gain functions associated with each node. Each of these functions computes the gain obtained by moving the given node to a specific subset. Moreover their method integrates others features. First, their gain function takes into consideration an intersubset cost metric. Second, their starting partition is not necessarily balanced, so they give precedence to the moves from large subsets to small ones. The complexity of one iteration is $O((p-1)M)$, with no bound for the number of iterations.

Also, Dutt [11] improved Kernighan and Lin's heuristic in the case of random sparse graphs. In this case, he proved that the search for optimal pairs can be done at low cost if the data structure has been built carefully. The complexity of one iteration is $O(M\log(N))$ but reduces to $O(\bar{d}M)$ for graphs such that the average degree $\bar{d}$ is less than $O(\log(N))$. Again, there is no bound for the number of iterations.

5.2 Balancing and reshaping the subsets

We now present our postprocessing heuristic which we named a *retrofitting* method after the term used by engineers when bracing existing structures. This method was first presented in [10] and has been naturally suggested by numerous numerical experiments conducted with GP. This heuristic is made of three steps. Steps 1 and 3 are identical *reshaping* steps while step 2 is a *balancing* step.

The reshaping step tries to redesign the outlines of subsets by deleting the *excrescences*, that is nodes which are attached to the subset to which they belong by a single edge. From a geometric point of view, by reassigning those nodes to neighboring subsets and by iterating the process until no more excrescences remain, the shape of the subsets is improved. Note that these boundary nodes are transfered to the subset which holds the highest number of its neighbors. Moreover, the value of m_0 can be improved but cannot deteriorate. Indeed, the number of edges that are cut from a vertex v goes from $d(v)-1$ (by definition of an excrescence) to $d(v)-d_{V_k}(v)$, where V_k is the subset to which v is reassigned, and $d_{V_k}(v)\geq 1$.

It is not always possible to eliminate all the excrescences, so the reshaping step has to stop when the overall shape of the partition, i.e. the number of excrescences, does not seem to be improved over the iterations (in practice five iterations). Here, one iteration consists of spanning the whole set of nodes V and reassigning the excrescences.

As we previously remarked, because of the reassignment of nodes to neighboring subsets in order to preserve the connectivity of the subsets, the resulting partition provided by GP is not balanced in most of the cases. Thus the second step of the retrofitting method looks for balancing the subsets by moving nodes from large subsets to small ones. One iteration of this consists of three parts (a), (b) and (c). First, in part (a), the largest and smallest subsets are determined. Then nodes of the largest subset are reassigned to its smallest neighbor in part (b). Last, in part (c), nodes are reassigned to the smallest subset, coming from its largest neighbor.

For this balancing step we have two strategies. On the one hand, the goal of the first method LB1, presented in [10], is to decrease the value of σ, i.e. to balance the subsets, while trying to reduce m_0, the number of edge cuts. Let us detail parts (b) and (c) for LB1. For this method, a *single* node is chosen and moved to reduce m_0. Let us call V_i the subset from which a node v is taken and V_j the subset to which it is reassigned. Then v is selected among the nodes of V_i bordering V_j. It is the one which leads to the best partition in terms of edge cuts. In other words, it is a node v of V_i which satisfies $d_{V_j}(v)\neq 0$ and maximizes $\{d_{V_j}(v)-d_{V_i}(v)\}$. Moreover, before moving a node v from subset V_i one has to make sure that $V_i\setminus\{v\}$ remains connected. For that, Proposition 3 is used. If this is not the case, then another boundary node of V_i is selected. If no node of V_i bordering V_j satisfies this connectivity requirement, then another subset (V_j in part (b), V_i in part (a)) has to be chosen.

On the other hand, for the second method LB2, the selection of the nodes that are to be reassigned in parts (b) and (c) differs from LB1 in that a whole *front* of nodes is moved (following the main idea used for the GP heuristic). Thus, instead of reassigning a single boundary node, all

boundary nodes, i.e. all nodes v of V_i such that $d_{V_j}(v) \neq 0$, are moved. So, balancing prevails over minimizing the number of edge cuts. This set of boundary nodes corresponds exactly to $F(V_j, 1) \cap V_i$. Moreover, as in the previous method, one has to check that $V_i \setminus F(V_j, 1)$ is connected. This is done using Proposition 3. If this is not the case, then another subset has to be selected. Evidently, this process is valid only when the standard deviation $\sigma = \sqrt{\frac{1}{p}\sum_{i=1}^{p}(n_i - \bar{n})^2}$ decreases. If we use a prime ($'$) for new quantities, call δn the number of nodes of the front and let $\delta\sigma = \sigma' - \sigma$, we have

$$\begin{aligned}
\delta\sigma &= (n_i' - \bar{n})^2 + (n_j' - \bar{n})^2 - (n_i - \bar{n})^2 - (n_j - \bar{n})^2 \\
&= (n_i - \bar{n} - \delta n)^2 + (n_j - \bar{n} + \delta n)^2 - (n_i - \bar{n})^2 - (n_j - \bar{n})^2 \\
&= 2\delta n\{\delta n - (n_i - n_j)\}.
\end{aligned}$$

In other words, LB2 is used when $\delta n \leq n_i - n_j$.

For both balancing steps, the process is iterated until the standard deviation approaches the optimal standard deviation

$$\sigma_{opt} = \sqrt{\frac{1}{p}[((q+1)p - N)(q - x)^2 + (N - qp)((q+1) - x)^2]}$$

where $x = \frac{N}{p}$ and $q = \lfloor \frac{N}{p} \rfloor$[3], or does not decrease anymore (in practice after five iterations).

The complexity of one iteration of the reshaping step is $O(M)$. The complexity of one iteration of the balancing step (either LB1 or LB2) is $O(p)$ for part (a) and $O(m_i + m_j)$ for parts (b) and (c). Note that, as for the postprocessing steps described in paragraph 5.1, when the retrofitting method is performed, neither the number of iterations for the reshaping steps nor for the balancing step are bounded *a priori*. Nevertheless, it is expected that the number of balancing steps which are performed when using LB2 is much smaller than when using LB1.

6 Examples

Many numerical experiments have already been presented by the authors on physical meshes for the GP heuristic on the one hand [9] and GP with the retrofitting method using LB1 on the other hand [10]. In the second case, a number of comparisons were made with spectrally derived heuristics, in terms of edge cuts, load balancing and elapsed time, the latter reflecting the overall complexity of the methods. Based on these experiments, it appeared that GP with LB1 compared quite well with the other heuristics (much faster but producing slightly more edge cuts [10]). Nevertheless, in a few cases the balancing time was very large. Therefore the need to design a new balancing step (LB2), leading to a heuristic totally based on fronts, hopefully faster.

The meshes for which we compare the greedy heuristics have the following characteristics:

Annulus	$N = 8448,$	$M = 33024.$
Airfoil	$N = 258990,$	$M = 773168.$
Spheres	$N = 9020,$	$M = 59418.$
Cube	$N = 15625,$	$M = 186696.$

[3] $\lfloor . \rfloor$ stands for the lower integer part of a number.

Eppstein $N = 131137, \quad M = 392256.$

Ê

Annulus, Airfoil and Eppstein are 2-dimensional meshes, whereas Spheres and Cube are 3-dimensional. For more details, we refer the reader to [10]. Hereafter we list in 15 Tables the result of the comparison of GP with the retrofitting method using either LB1 or LB2 (called respectively GP_1 and GP_2). The parameters listed below are the *average standard deviation* σ/n (%), the percentage of edge cuts m_0/M and the elapsed time t in seconds (on a "Viking" SuperSparc processor).

Annulus

p	GP_1	GP_2
4	0.3	0.3
16	0	0
64	4.4	4.1
256	2.9	2.9

Table 1.1: σ/n (%).

p	GP_1	GP_2
4	2.63	2.64
16	5.83	5.83
64	11.72	11.85
256	24.75	24.79

Table 1.2: m_0/M (%).

p	GP_1	GP_2
4	0.5	0.4
16	0.3	0.2
64	0.7	0.4
256	0.5	0.4

Table 1.3: t (s).

Airfoil

p	GP_1	GP_2
16	0	0
64	6.7	6.7
256	4.3	3.0
1024	3.0	2.1

Table 2.1: σ/n (%).

p	GP_1	GP_2
16	1.00	1.00
64	1.88	1.92
256	4.08	4.11
1024	8.35	8.37

Table 2.2: m_0/M (%).

p	GP_1	GP_2
16	300.0	9.2
64	46.0	12.0
256	18.0	14.0
1024	16.0	12.0

Table 2.3: t (s).

Spheres

p	GP_1	GP_2
4	0	0
16	0	0.1
64	0.8	0.4
256	1.8	1.6

Table 3.1: σ/n (%).

p	GP_1	GP_2
4	8.03	8.03
16	18.30	18.29
64	29.81	29.83
256	47.51	47.54

Table 3.2: m_0/M (%).

p	GP_1	GP_2
4	0.3	0.2
16	0.3	0.3
64	0.8	0.5
256	0.4	0.4

Table 3.3: t (s).

Cube

p	GP_1	GP_2
4	0	0
16	0.1	0.1
64	0.2	0.2
256	0.5	0.5

Table 4.1: σ/n (%).

p	GP_1	GP_2
4	7.43	7.43
16	16.35	16.37
64	27.22	27.22
256	46.06	46.05

Table 4.2: m_0/M (%).

p	GP_1	GP_2
4	0.7	0.6
16	2.8	2.4
64	0.8	0.8
256	1.0	0.9

Table 4.3: t (s).

Eppstein

p	GP_1	GP_2
16	0	0
64	3.5	2.1
256	1.1	1.1
1024	1.4	1.1

Table 5.1: σ/n (%).

p	GP_1	GP_2
16	1.35	1.35
64	2.96	2.96
256	6.01	6.00
1024	12.03	12.02

Table 5.2: m_0/M (%).

p	GP_1	GP_2
16	7.7	4.3
64	6.2	5.9
256	3.9	3.5
1024	4.9	4.4

Table 5.3: t (s).

Based on this series of numerical experiments, we can make several remarks. First, the average load balancing is improved when using LB2. Second, the number of edge cuts m_0 is only marginally higher for this balancing step, although it does not take into account the optimization of m_0. Third, elapsed times are always smaller and, in some cases, LB2 produces results equivalent to those obtained with LB1 much faster. Moreover, for a given mesh, there are no large fluctuations in terms of elapsed time for different values of the number of subsets (see the Airfoil). Finally, let us mention that if we divide the number of edge cuts by the average of the elapsed time (for a given mesh and different values of p), then we have, for the partitioning heuristic GP_2, the following results[4]:

Annulus $M/t \approx 9.10^4$ edges/s.
Airfoil $M/t \approx 7.10^4$ edges/s.
Spheres $M/t \approx 17.10^4$ edges/s.
Cube $M/t \approx 16.10^4$ edges/s.
Eppstein $M/t \approx 9.10^4$ edges/s.

7 Conclusion

In this paper, we have presented a heuristic which approximates the p-edge partitioning problem for large sparse graphs G with a connectivity constraint on the subsets. It is decomposed into two parts: the computation of an initial partition and an iterative retrofitting process. We proved the finiteness and correctness of the initial step and also showed that its complexity is $O(M)$, where M is the number of edges of the graph, by using front-based tools. For the retrofitting method, we designed a new balancing step, also based on front techniques, in order to reduce the number of iterations of the method. Numerically, we verified on various examples that the new retrofitting process leads to reduced elapsed time to compute the partition, and that these elapsed times are linear with respect to M.

[4]For GP alone, M/t is in the order of 18.10^4 edges/s.

References

[1] C.J. Alpert and A.B. Kahng. Multi-way system partitioning via geometric embeddings, spacefilling curves and dynamic programming. *Personal communication UCLA-CS dept.*, 1994.

[2] S.T. Barnard and H.D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. Technical report, NASA Ames Research Center, RNR-092-033, 1992.

[3] E.R. Barnes. An algorithm for partitioning the nodes of a graph. *SIAM J. Alg. Disc. Meth.*, vol 3, n° 4, pp 541-550, 1982.

[4] E.R. Barnes, A. Vannelli, and J.Q. Walker. A new heuristic for partitioning the nodes of a graph. *SIAM J. Disc. Meth.*, vol 1, n° 3, pp 299-305, 1988.

[5] C. Berge. *Graphes.* Gauthier–Villars, Paris, 1983.

[6] T.N. Bui, S. Chaudhuri, F.T. Leighton, and M. Sipser. Graph bisection algorithm with good average case behavior. *Combinatorica*, pp 171-191, 1987.

[7] N. Chistofides and P. Brooker. The optimal partitioning of graphs. *SIAM J. on Applied Mathematics*, vol 30, n° 1, pp 55-69, 1976.

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

[9] P. Ciarlet, Jr and F. Lamour. An efficient low cost greedy graph partitioning heuristic. Technical report, UCLA, CAM 94-1, 1994.

[10] P. Ciarlet, Jr and F. Lamour. Recursive partitioning methods and greedy partitioning methods: a comparison on finite element graphs. Technical report, UCLA, CAM 94-9, 1994.

[11] S. Dutt. New faster kernighan-lin graph partitioning algoritms. *IEEE/ACM International conference on CAD-93*, pp 370-377, 1993.

[12] C. Farhat. A simple and efficient automatic FEM domain decomposer. *Computers and structures*, vol 28, *n°* 5, pp 579-602, 1988.

[13] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. *Proceedings of the 19th IEEE Design Automation conference*, IEEE, pp 175-181, 1982.

[14] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23, (98), pp 298-305, 1973.

[15] L.R. Foulds. *Graph Theory applications.* Springer–Verlag, 1992.

[16] L. Hagen and A.B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE transactions on computer-aided design*, vol 11, n° 9, pp 1074-1085, 1992.

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

[18] B. Hendrickson and R. Leland. Multidimensional spectral load balancing. Technical report, SAND 93-0074, 1993.

[19] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical report, SAND 93-1301, 1993.

[20] B.W. Kernighan and S. Lin. An efficient heuristic for partitioning graphs. *The Bell System Technical Journal*, vol 49, n° 2, pp 291-307, 1970.

[21] J.W.H. Liu. A graph partitioning algorithm by node separators. *ACM Transactions on Mathematical Software*, vol 15, n° 3, pp 198-219, 1989.

[22] G. L. Miller, S-H Teng, W. Thurston, and S.A. Vavasis. *Automatic mesh partitioning*, pages 57–84. Springer-Verlag, 1993. Graph theory and sparse matrix computations.

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

[24] C. Roucairol and P. Hansen. Problème de la bipartition minimale d'un graphe. *RAIRO*, vol 21, *n°* 4, pp 325-348, 1987.

[25] H.D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, vol 2, n° 2/3, pp 135-148, 1991.