A new method, the fusion fission, for the relaxed k-way graph partitioning problem, and comparisons with some multilevel algorithms

Charles-Edmond Bichot

A new Method, the Fusion Fission, for the relaxed k-way graph partitioning problem, and comparisons with some Multilevel algorithms.*

Charles-Edmond Bichot
LOG (DSNA — ENAC), 7 av. Edouard Belin 31 000 Toulouse, France
bichot@recherche.enac.fr

Abstract

In this paper a new graph partitioning problem is introduced, the relaxed k-way graph partitioning problem. It is close to the k-way, also called multi-way, graph partitioning problem, but with relaxed imbalance constraints. This problem arises in the air traffic control area. A new graph partitioning method is presented, the Fusion Fission, which can be used to resolve the relaxed k-way graph partitioning problem. The Fusion Fission method is compared to classical Multilevel packages and with a Simulated Annealing algorithm. The Fusion Fission algorithm and the Simulated Annealing algorithm, both require a longer computation time than the Multilevel algorithms, but they also find better partitions. However, the Fusion Fission algorithm partitions the graph with a smaller imbalance and a smaller cut than Simulated Annealing does.

keywords

Graph partitioning, Multilevel, Metaheuristics, Fusion Fission

1 Introduction

Graph partitioning is an important problem in computer science that have applications on many areas like image segmentation, VLSI design, task scheduling and parallel computing. The task is to divide the set of vertices of a graph into some parts which can be subjects to some constraints, while minimizing an objective function.

The graph partitioning problem is known to be NP-complete, even if the graph is not weighted [GJS76]. This means that there is essentially only one way known to prove that a given solution is optimal: to list all other solutions with their associated cost returned by the objective function. Therefore, many efficient heuristics method have been developed to find reasonably good partitions in an appropriate time.

Researchers have developed many algorithms for graph partitioning. Depending on what kind of input they require, algorithms can be divided into two categories: local and global methods.

*accepted for publication the 1st December 2006 in the Journal of Mathematical Modelling and Algorithms (JMMA).

Local improvement algorithms take as input a partition of a graph and decrease the cost returned by the objective function. Specific local methods have been developed to resolve the graph partitioning problem. One of the earliest, which is very powerful and always often used, is the method proposed by Kernighan and Lin [KL70]. This method can be usefully improved by the Fiduccia and Mattheyeses algorithm [FM82], which reduces its complexity. A second efficient specific local method is proposed by Diekmann, Monien and Preis [DMP95]. It is based on the notion of k-helpful sets. Some metaheuristics, which are general local methods, have been adapted to resolve the graph partitioning problem: Simulated Annealing [JAMS89], Ants Colonies [KLS97, LG99] and Genetic Algorithms [TB91, Gre01].

Global methods start with the graph not partitioned. They are often used in combination with some local improvement methods. Geometric methods can be classified in a first category. To be used, each node of the graph to be partitioned must have a geometric location. Another category of global methods includes hierarchical methods. It is based on data agglomeration. Third, spectral method are based on the computation of some eigenvectors of the matrix of the graph. The very often use Multilevel method is also a global method. It is based on a coarsening phase, a partitioning phase and an uncoarsening phase.

A new partitioning method is presented in this paper. It is based on nuclear techniques and particularly on the fusion and the fission process. This new partitioning method, the Fusion Fission method, can be viewed as a metaheuristic. Indeed, it can be applied to many subjects, but only its application on the graph partitioning problem is presented in this article.

The Fusion Fission method is divided into two parts : an initialization and an optimization process. The initialization is an agglomerative process used to create a valid initial solution, and the optimization is a local combinatorial search.

The remainder of the paper is organized as follows. Section 2 defines the general graph partitioning problem. Section 3 describes the bases of the Fusion Fission method and describes the percolation method. Section 4 presents the algorithm of Fusion Fission used to solve the relaxed k-way graph partitioning problem. Section 5 describes the Multilevel method and the Simulated Annealing algorithm for the graph partitioning problem. Section 6 presents an application of the relaxed k-way partitioning problem on the Air Traffic Control and compares some classical libraries used in graph partitioning to the Fusion Fission and Simulated Annealing algorithms.

2 Graph partitioning

The general problem of graph partitioning is to partition the vertices of a graph G into several parts respecting constraints, while minimizing an objective function. Depending on goals, constraints are different. A classical graph partitioning problem consists to fix the number of partitions, to find parts with roughly the same weight, and to decrease the number of edges connecting vertices in different parts. This is the problem of VLSI design or parallel processing partitioning and this problem is named the *k-way graph partitioning problem* (it is also called multi-way graph partitioning problem). A second problem, *the image segmentation problem*, consists only in decreasing an objective function called the normalized cut function, for a variable number of partitions, with no constraint on parts.

The graph partitioning problem can be formerly defined as follows: Given a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. If edges or vertices

are not weighted then set all of them to an unit weight. $\forall v \in V, w(v)$ is the weight of the vertex v and $\forall e = (u,v) \in E, w(u,v)$ is the weight of the edge e. Find a partition P_k of k subsets $V_1, \ldots, V_k$ of V such that:

- $\bigcup_{i=1}^{k} V_i = V$
- $\forall i, j \in \{1, \ldots, k\}, i \neq j, V_i \cap V_j = \emptyset$
- $\forall i \in \{1, \ldots, k\}$, the constraint $C(V_i)$ is true
- The objective function $obj(P_k)$ is minimized

Let *Cut* be the cut between two distinct parts (*i.e.* the sum of the weights of the edges between part V_i and part V_j for $i \neq j$):

$$Cut(V_i, V_j) = \sum_{u \in V_i, v \in V_j} w(u,v)$$

Let W be the weight of a part V_i (*i.e.* the sum of the weights of the vertices of part V_i):

$$W(V_i) = \sum_{v \in V_i} w(v)$$

The three classicals graph partitioning problems are:

- The bisection graph partitioning problem. It is a k-way graph partitioning problem where $k = 2$.
- The k-way graph partitioning problem is defined with the constraint C_k:

  $$\forall i, W(V_i) < Imbalance * \left\lceil \frac{W(V)}{k} \right\rceil$$

  where the function $f : x \rightarrow \lceil x \rceil$ returns the smallest integer greater than x, and *Imbalance* is low (from 1.0 to 1.1). The k-way graph partitioning problem uses the objective function:

  $$Cut(P_k) = \sum_{i<j} Cut(V_i, V_j)$$

  (*i.e.* the sum of the weights of edges cut by the partition).
- The normalized cut partitioning problem [SM00] is defined with no constraint on the size of parts, and with the objective function:

  $$Ncut(P_n) = \sum_{i} \frac{Cut(V_i, V - V_i)}{Cut(V_i, V)}$$

The weight imbalance of a partition, w_{imb}, is defined as the maximum subdomain weight, W_{max}, divided by the optimal weight of a part, W_{opt}:

$$W_{max} = \max_{i \in \{1,\ldots,k\}} W(V_i) \quad \text{and} \quad W_{opt} = \left\lceil \frac{W(V)}{k} \right\rceil$$

Thus:

$$w_{imb} = \frac{W_{max}}{W_{opt}}$$

The vertex imbalance v_{imb} is defined as the maximum subdomain cardinal, C_{max}, divided by the optimal, C_{opt}:

$$C_{max} = \max_{i \in \{1,\dots,k\}} |V_i| \quad \text{and} \quad C_{opt} = \left\lceil \frac{|V|}{k} \right\rceil$$

Thus:

$$v_{imb} = \frac{C_{max}}{C_{opt}}$$

Then the *Imbalance* constraint is equal to w_{imb} or v_{imb}, depending if the graph is weighted or not.

The purpose of this paper is to solve a new graph partitioning problem which is between the k-way partitioning problem and the normalized cut partitioning problem. The problem deals with relaxed k-way partitioning problem constraints:

$$\forall i \in \{1,\dots,k\}, W(V_i) < \left\lceil \frac{W(V)}{k} \right\rceil * w_{imb}$$

and constraints on parts cardinals:

$$\forall i \in \{1,\dots,k\}, |V_i| < \left\lceil \frac{|V|}{k} \right\rceil * v_{imb}$$

where the imbalances v_{imb} and w_{imb} are high: from 1.5 to 3.0. The objective function can be the *Cut* function or the *Ncut* function, but the article only focus on the *Ncut* function. This problem was named the *relaxed k-way partitioning problem*.

While the literature on the k-way partitioning problem [KK98b, LG99, SWC04] and on the normalized cut partitioning problem [SM00, DGK04] is extensive, it is low on the relaxed k-way partitioning problem. But, as shown in section 6, this problem arises in real life applications. However, there is actually no method to resolve this specific problem with all of its constraints. Thus a new graph partitioning method, called Fusion Fission, was created to resolve this problem.

3 The Fusion Fission bases

As many metaheuristics, the Fusion Fission method takes origin in the real life. The Fusion Fission method comes from the nuclear process. The nuclear process can be viewed as a matter reorganization in an optimization process which tends to create atoms of great internal cohesion. In the nature, the atom with the greatest internal cohesion is the iron atom *Fe*, with 56 nucleons in a range of 2 to 235 for other atoms. We can imagine that a reorganization of the nucleons of the atoms naturally tends to create iron atoms, if the number of nucleons and the sort of nucleons allows it.

The graph partitioning problem has some similarities with the nuclear process. The objective of the graph partitioning problem is to find a low energy organization of the parts of a graph. Thus, it is easy to compare atoms with parts and nucleons with vertices. Then, like in the nature, the process to find an organization of low energy consists in merging or splitting atoms. So, parts of the partition are successively merged or split by the Fusion Fission process.

The Fusion Fission method consists in repeating a perturbation process of a partition till a stop condition is reached. The perturbation process starts with a partition of the graph and successively applies fusion or fission to the partition.

A nuclear reaction can only be created in a very high temperature plasma. Thus the Fusion Fission method includes a "temperature" which is used to channel the process. The temperature is used to stop the perturbation process after a definite number of pass.

In the nature, when a fusion or a fission is done, some nucleons are ejected by the process. These nucleons join another atoms, or make other fissions if they have a very high energy. The number of ejected nucleons is know with a certain probability, and the "rules" of ejected nucleons for the reactions can be written. Then some "rules" are defined to eject randomly in a fusion or in a fission some vertices of its part. These rules are automatically adjusted during the execution of the algorithm by a self-learning function. A rule is just a probability function to eject a certain amount of vertices of the atom. Vertices which have a proportionally low connection[1] with the other vertices of the same part are ejected first.

The process of fusion consists just in merging two parts. The new part is the contraction of the two former parts minus the random number of vertices "ejected" by the rule. Then, if it is possible, these vertices are merged with a different part with which they are the most connected.

The process of fission is much more complex. To split a part into two new parts, a bisection method must be used. There is a great number of bisection methods. An agglomerative method based on the percolation process was chosen in this paper. The algorithm of this method is presented in section 3.1. But before splitting the part, like in the fusion process, some vertices can be "ejected" of the part. Then these vertices can be merged with a partition with which they have big connection, or they can split other parts, in a chained fission reaction.

3.1 The percolation method

The percolation algorithm presented in this section was used to make a bisection of a partition's part in the Fusion Fission algorithm. The percolation is defined as the spreading of a liquid flow through a porous substance. For example, the coffee drink is extracted from coffee by the percolation process. The principle of percolation is as follow: the liquid starts on a place, and then spreads gradually all over this place. Formerly, the percolation is the deterministic move of a fluid through a probabilistic matter.

The percolation process was adapted to the bisection problem. To cut a graph in two parts, two colored percolations were used. Let be V_i the set of vertices of the graph to be cut. Select randomly two vertices $c_1, c_2 \in V_i$. These two vertices are the start points of the two colored liquids. Each color represents one of the new parts $V_{i,j}, j \in \{1,2\}$. The algorithm progresses step by step. A vertex $v \in V_i$ is added to a part $V_{i,j}$ — then it is colored — if $\forall k \in \{1,2\}, k \neq j$:

$$link(v, V_{i,j}) > link(v, V_{i,k})$$

with :

$$link(v, P_j) = \sum_{e \in \ path \ from \ c_j \ to \ v} \frac{w(e)}{2^d}$$

where d is the number of vertices between v and c_j. The distance to the center of a partition is arbitrary represented by the division by 2^d.

[1] The connection between a vertex and its part is the sum of the weight of the edges connecting this vertex to vertices of the same part

All the links are recomputed at each step. A link is calculated as follow: let $v \in V_i$, let $N(v)$ its neighborhood[2]. For each vertex in $N(v)$ which has already a link calculated, calculate the new link with v. The new link of v is the lowest link calculated before. The algorithm stops when no vertex moves from a part to the other.

The percolation algorithm is described in algorithm 1.

Algorithm 1 Percolation

```
let vertices_alone = V_i
let map_of_links = ∅
while vertices_alone ≠ ∅ and some vertices are moved do
  for all part P_j, j ∈ {1,2} do
    for all v ∈ P_j do
      let neighborhood = {v ∈ V_i, ∃u ∈ V_i, (u,v) ∈ E};
      let link_v = find v in map_of_links;
      for all v_n ∈ neighborhood do
        let link_{v_n} = find v_n in map_of_links;
        let new_link = w(v,v_n) + link_{v_n}/2;
        if new_link > link_{v_n} then
          map_of_links = add v_n new_link map_of_links;
          if P_j ≠ part of v_n then
            move v_n in P_j;
```

4 The Fusion Fission algorithm for the relaxed k-way partitioning problem

This paper only focus on the relaxed k-way partitioning problem with *Ncut* as an objective function. As we say in section 2, the problem consists in partitioning a graph $G = (V,E)$ into a fixed number of parts k, while decreasing the *Ncut* objective function and subject to constraints on parts $V_i, \forall i \in \{1,\ldots,k\}$: $W(V_i) < \left\lceil \frac{W(V)}{k} \right\rceil * w_{imb}$ and $|V_i| < \left\lceil \frac{|V_i|}{k} \right\rceil * v_{imb}$, with w_{imb} and v_{imb} between 1.5 and 3.0.

All partitioning methods work with a constant number k of parts. But the Fusion Fission method is based on the change of the number of parts of the partition. In its current form, the Fusion Fission method can not find solutions with a low imbalance. Indeed, it can find some solutions for different number of parts, which can be very useful for some applications like image segmentation, where this number of parts is unfixed.

The results returned by the *Ncut* objective function depend on the number of parts of the partition to evaluate. The lowest values of the *Ncut* objective functions are found for a graph which is not partitioned. And the values returned by *Ncut* are increasing with the number of parts. Because of the change of the number of parts of partitions found during the process of Fusion Fission, the objective function must be adapted. The adaptation of the objective function must be neutral for partitions with nearly k parts and penalize partitions with a number of parts $n \ll k$ or $n \gg k$.

[2]The neighborhood of a vertex is the set of vertices connected to this vertex by an edge

A correction function was experimentally design:

$$f_{correction}(partition) = \left(\frac{k - |partition|}{3}\right)^2 + 1$$

with $|partition|$ the number of parts of the partition. Then the cost function

$$energy = f_{correction} * Ncut$$

is used in the Fusion Fission algorithm.

The Fusion Fission algorithm used is presented in the algorithm 2. The algorithm has a main loop, which stops when a stop condition is reached. The condition, *stop_condition*, used is a calculation time condition, which means that the loop ends after a minimum duration time. Two identifiers are initialized before the loop with the functions *rules_initialization* described in subsection 4.4 and *partition_initialization* described in subsection 4.5. Each main loop pass is initialized by setting the current partition to the best partition found before. Then there is a second loop of NB_OF_PASS pass. For each pass, a part *part* is randomly chosen among all parts of the current partition by the *choose_partition* function. Then the choice between a fusion and a fission of *part* is made by the *fusion_choice* function. This function is described in subsection 4.1. This choice results to the use of the function *fusion* described in subsection 4.2 or to the use of the function *fission* described in subsection 4.3. After that the part has been split by the fission, or fused to another part by the fusion, rules which have served to the fusion or to the fission process are updated. Then the best partition found is updated if necessary. A new pass of the second loop can start with the new partition computed.

Algorithm 2 Fusion Fission

```
let rules = rules_initialization();
let best_partition = partition_initialization();
while stop_condition() do
  let current_partition = best_partition;
  for temperature = 1 to NB_OF_PASS do
    let part = choose_partition(current_partition);
    let new_partition = current_partition;
    if fusion_choice(part, current_partition, temperature) then
      new_partition ⇐ fusion(part, current_partition, rules);
    else
      new_partition ⇐ fission(part, current_partition, rules);
    rules ⇐ update_rules(rules);
    if energy(new_partition) < energy(best_partition) then
      best_partition ⇐ new_partition;
    current_partition ⇐ new_partition;
```

4.1 Process of choice between fusion and fission

The choice between a fusion or a fission is made by the function of choice *fusion_choice*. A lot of choice function were tested. One of the simplest, which is also one of the best, is presented here. This function depends on the cardinal of the part: $|part|$, and of the

"temperature", *i.e.* the current number of pass of the second loop (see algorithm 2). It uses the average number of vertices in a part: $\frac{|V|}{k}$, and the total number of pass of the second loop NB_OF_PASS. The function $fusion\_choice$ compares a random number to the function $choice(|part|, temperature)$. This function *choice* uses the function $\alpha(temperature)$ which uses two parameters, MULT and ADD:

$$\alpha(temparature) = MULT * \frac{NB\_OF\_PASS - temperature}{NB\_OF\_PASS} + ADD$$

$$choice(|part|, temperature) = \alpha(temparature) * \left(|part| - \frac{|V|}{k} \right) + \frac{1}{2}$$

The *choice* function returns a real number which is $\frac{1}{2}$ if *part* is perfectly balanced. This number is bigger if the partition is bigger, and smaller in the other case.

The function $fusion\_choice$ can now be defined. For each function call, a random number $r \in [0;1]$ is chosen. The function $fusion\_choice$ returns the boolean: $r > choice(|part|, temperature)$. Thus, if r is greater than *choice*, there is a fusion, else, there is a fission.

The temperature is used to manage the process as it is done in a real nuclear reaction. Higher is the temperature of the plasma, higher is the probability to have a fusion or a fission. Weight of the part could be used instead of its cardinality. But the use of part's cardinality allows to control the imbalance of the number of vertices of the partition.

4.2 Fusion

The fusion algorithm which merged a part, called *part*, with another part of the partition, called *partition* or P_k, is described in algorithm 3. The algorithm starts to choose the most connected neighbor part of *part* in the partition with the function *choose_neighbor*. The function *choose_neighbor* computes for each part of the partition P_k, the sum of the weight of the edges which join this part with *part*, and returns the part of maximal sum. With the notations of section 2, $choose\_neighbor(P_k)$ finds the part V_l which verify:

$$Cut(V_l, part) = \max_{V_j \in P_k} Cut(V_j, part)$$

This second part, called *second_part*, is fused with *part* to create a new part called *new_part*. Some vertices are ejected of *new_part* with the help of the rules to make a list of vertices, as describe in subsection 4.4. Then, in *partition*, *part* and *second_part* are replaced by *new_part*. After updating *partition*, all vertices of the list of vertices are added to the part from which they are the most connected. Formerly, *most_connected_part* searches the part V_m which verify:

$$Cut(V_m, \{vertex\}) = \max_{V_j \in P_k} Cut(V_j, \{vertex\})$$

Finally, *partition* is updated with the new membership of *vertex* in the partition.

4.3 Fission

The fission algorithm of a part into two new parts is described in algorithm 4. The algorithm starts to eject some vertices of the initial part called *part* in the same way

Algorithm 3 $fusion$(part, partition, rules)

```
let second_part = choose_neighbor(partition);
let new_part = fuse(part, second_part);
let list_of_vertices = eject_vertices(new_part, rules);
partition ⇐ remove(partition, [part; second_part]);
partition ⇐ add(partition, [new_part]);
for all vertex ∈ list_of_vertices do
    let con_part = most_connected_part(partition, vertex);
    con_part ⇐ add_vertex(con_part, vertex);
    partition ⇐ replace(partition, con_part);
```

than in the fusion algorithm. Then *part* is split in two new parts called $first\_part$ and *second_part*, by the percolation algorithm as described in section 3.1. The partition is updated by removing *part* and adding the two new parts. After that, for each vertex of the list $list\_of\_vertices$, the part of higest energy *e_part* is chosen by the function *highest_energy_part*. Depending on the energy of *e_part*, the vertex may be added to a *e_part* like in the fusion algorithm, or may split the part *e_part* in two. Let the energy of a part V_i of the partition P_k be:

$$energy(V_i) = \frac{Cut(V_i, V - V_i)}{W(V_i)}$$

The function *highest_energy_part* returns the part *e_part* which verifies:

$$energy(e\_part) = \max_{V_j \in P_k} energy(V_j)$$

Thus, the part *e_part* is the part of highest energy of the partition. After having found *e_part*, its energy is compared with the average energy of a part of the partition. The *average_energy* function is computed as follows:

$$average\_energy(P_k) = \sum_{V_j \in P_k} \frac{energy(V_j)}{k}$$

If the energy of *e_part* is big enough, a chained fission process begins. The chained fission process simply consists in splitting the part *e_part* in two and to update $partition$ like it is done at the beginning of the fission algorithm. Else, if the energy of *e_part* is not big enough, the vertex is added to its most connected part like the fusion algorithm does. CHAINED is the parameter which controls the number of occurrences of the chained fission process.

4.4 Rules process

As we have seen before, rules are used to eject some vertices of a part V_i. The main idea of finding a rule is to find a percentage of vertices of V_i to be ejected. The rules are split in two sections, one for the fusion process and one for the fission process. And then, each of this two sections are split into 5 subdomains. A rule is chosen regarding the cardinal of the part $|V_i|$ and regarding if this is a fusion or a fission. Let $Opt_v = \frac{|V|}{k}$ be the average number of vertices in a part. A fusion rule, like a fission rule, is chosen as explained by figure 1. Each of these 5+5 rules, is an array of the probabilities to

Algorithm 4 *fission*(part, partition, rules)

```
let list_of_vertices = eject_vertices(part, rules);
let (first_part, second_part) = bisection(part);
partition ⇐ remove(partition, part);
partition ⇐ add(partition, [first_part; second_part]);
for all vertex ∈ list_of_vertices do
  let e_part = highest_energy_part(partition);
  if energy(e_part) ≤ CHAINED * average_energy(partition) then
    e_part ⇐ add_vertex(e_part, vertex);
    let (first_part, second_part) = bisection(e_part);
    partition ⇐ remove(partition, [e_part]);
    partition ⇐ add(partition, [first_part; second_part]);
  else
    let con_part = most_connected_part(partition, vertex);
    con_part ⇐ add_vertex(con_part, vertex);
    partition ⇐ replace(partition, con_part);
```

eject some vertices of the part V_i. The sum of these probabilities must be strictly equal to one. A probability r is selected randomly in $[0;1]$. r belongs to a range of the sum of the probabilities of a rule. Let's define j as the element number of the lower bound of this range. For all array of probabilities $rule[j]$, there is an array of percentages $percent[j]$. There are percentages of the number of vertices which can be ejected of the part V_i. Then, the percentage of vertices to be ejected is the element number j of the percentage array. If p is the percentage of vertices to be ejected, the number of vertices is the lower bound $\lfloor p * |V_i| \rfloor$.

This process is illustrated by an example. Let's define V as a set of vertices of cardinal 1000 and $k = 20$ as the number of parts and $|V_i| = 42$ the cardinal of V_i. Assume that V_i is in the fusion process. Thus V_i is *new_part* in the algorithm 3. Because

$$0.7 * Opt_v = 35 < |V_i| = 42 \leq 0.9 * Opt_v = 45$$

the rules in the fusion rules is $rule\_fusion[1]$ (see figure 1). Assume that

$$rule\_fusion[1] = \{0.7; 0.2; 0.06; 0.04\}$$

$$percent[1] = \{0.0; 0.02; 0.05; 0.1\}$$

Assume that the random number found is $r = 0.945$. Then, to search the probability of $rule\_fusion[1]$, we calculate:

$$0.7 < 0.7 + 0.2 = 0.9 \leq r < 0.9 + 0.06 = 0.96 < 0.96 + 0.04 = 1$$

Thus, the probability of $rule\_fusion[1]$ randomly chosen is

$$rule\_fusion[1][3] = 0.06$$

Also, the percentage of vertices to be ejected is

$$percent[1][3] = 0.05$$

Finally, the number of vertices ejected of V_i is

$$\lfloor percent[1][3] * |V_i| \rfloor = 2$$

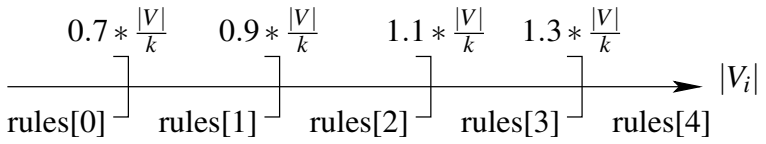

Figure 1: A rule is chosen depending on the number of vertices of the part V_i and how far is $|V_i|$ from the average number of vertices in a part: $Opt_v = \frac{|V|}{k}$

The rules are initialized identically for each fusion rule and for each fission rule. It could be surprising to initialize them identically, but it is simpler and rules automatically adjust themselves. After initialization

$$\forall j \in \{1,\ldots,4\}, rules\_fusion[j] = rules\_fusion[0]$$

$$rules\_fission[j] = rules\_fission[0]$$

In the same way, after initialization:

$$\forall j \in \{1,\ldots,4\}, percent[j] = percent[0]$$

The process which automatically adjusts rules consists in changing the rule array $rule\_fusion[j]$ or $rule\_fission[j]$ after each fusion or fission. Assume that we have a fusion (it is the same way for fission), that $rule\_fusion[j][m]$ is selected. If the new partition after the fusion has a lower energy than before, then $rule\_fusion[j][m]$ is upgraded and its new value is

$$rule\_fusion[j][m] = rule\_fusion[j][m] + RULE\_ADJUSTMENT$$

Because

$$\sum_{n=0}^{q} rule\_fusion[j][n] = 1$$

(with q the length of the array $rule\_fusion[j][m]$)
$\forall n \in \{0,\ldots,q\}, n \neq m$:

$$rule\_fusion[j][n] = rule\_fusion[j][n] - \frac{1}{q} RULE\_ADJUSTMENT$$

Else, if the new partition after the fusion has a higher energy than before, then

$$rule\_fusion[j][m] = rule\_fusion[j][m] - RULE\_ADJUSTMENT$$

$\forall n \in 0,\ldots,q, n \neq m$:

$$rule\_fusion[j][n] = rule\_fusion[j][n] + \frac{1}{q} RULE\_ADJUSTMENT$$

Rules presented here are highly customizable. Of course, new rules process can be used. We are exploring some new rules process, and how to simplify them.

4.5 Fusion Fission region growing initialization

The Fusion Fission method can be adapted to make a region growing algorithm. The initialization is roughly a reduction of the Fusion Fission process. The initialization starts with a graph where each vertex is a partition. So, the number of partitions and the number of vertices are the same. Then, the cost value of the initial partition is maximal. The first goal of initialization is to group vertices to obtain a near k-partition. Secondly, the initialization stops when the cost value of the current partition is lower than the parameter INIT.

The initialization process is described in algorithm 5. It is composed of a main loop which ends when the number of parts of the current partition is around k (plus or minus 2), and when the energy of this partition is under the threshold INIT. Each pass in the loop is composed of three steps. The first step consists in choosing randomly a part with the function *choose_partition* already used in the Fusion Fission algorithm 2. In the second step, a fusion or a fission is selected by the $fusion\_choice$ function. Then the fusion or the fission is done. If it is a fusion, a second part is chosen, which is fused with the first one. And after that, some vertices are ejected with the help of rules (see subsection 4.4), and the current partition is updated. Else, if it is a fission, some vertices are immediately ejected with the help of rules (see subsection 4.4), and after that, the part is split by the percolation algorithm (see subsection 3.1), and the partition is updated. In the third step, each vertex ejected before is moved into its most connected part as it is done in the fusion process 4.2.

Algorithm 5 Fusion Fission initialization

```
let partition = graph;
while not (|partition| ≈ k and energy(partition) < INIT) do
  let part = choose_partition(partition);
  let list_of_vertices = ∅;
  if fusion_choice(part, partition) then
    let second_part = choose_neighbor(partition);
    let new_part = fuse(part, second_part);
    list_of_vertices ⇐ eject_vertices(new_part, rules);
    partition ⇐ remove(partition, [part; second_part]);
    partition ⇐ add(partition, [new_part]);
  else
    list_of_vertices ⇐ eject_vertices(part, rules);
    let (first_part, second_part) = bisection(part);
    partition ⇐ remove(partition, part);
    partition ⇐ add(partition, [first_part; second_part]);
  for all vertex ∈ list_of_vertices do
    let con_part = most_connected_part(partition, vertex);
    con_part ⇐ add_vertex(con_part, vertex);
    partition ⇐ replace(partition, con_part);
```

5 Another algorithms for the relaxed k-way partitioning problem

5.1 The Multilevel method

The Multilevel method is applied to the graph partitioning problem since the 1980's. This is a well studied method [HL95b, AHK97, KK98b, WCM00]. It is even sometimes described as a metaheuristic [Wal04].

The Multilevel method is a three phase method. First, the graph is coarsened. Secondly, the smallest graph is partitioned. Third, the graph is uncoarsened to construct the partition of the original graph. Let us describe more precisely the method:

The coarsening phase: the aim is to make a contraction of a large number of edges that are well dispersed throughout the graph. During this step, some couple of vertices (a and b) joined by an edge are merged. The resulting new vertex has as weight value the sum of the weights of a and b. Each edge of adjacent vertices of a and/or b, have a new weight which is the sum of the weight between the adjacent vertex and the two vertices a and/or b.

The partitioning phase: during this step, different partitioning algorithms can be used. Hendrickson and Leland use a spectral method which uses the eigenvectors of the Laplacian matrix. Another method consists in coarsening the graph till it has k vertices, thus the partition is trivial.

The uncoarsening phase: because each vertex in a coarse graph is simply the union of vertices from the larger graph, this step is trivial. But the best partition of the coarse graph may not be optimal for its uncoarsened counterpart. Therefore a local refinement method can be periodically applied during this uncoarsened phase, such as the Kernighan-Lin algorithm [KL70] or the helpful-sets algorithm [DMP95].

5.2 Simulated Annealing

Metaheuristics are useful tools to resolve problems for which specific algorithms have not been designed, or are not powerful. One of the earliest but always very powerful metaheuristic, is the Simulated Annealing. The Simulated Annealing algorithm presented in this paper is based on the article of David Johnson, Cecilia Aragon, Lyle McGeoch and Catherine Schevon [JAMS89]. The authors present in this article an application of Simulated Annealing to the unweighted bisection graph partitioning problem. Thus, to resolve the relaxed k-way partitioning problem some details were changed in their algorithm.

The Simulated Annealing method is not explained in this article. [KGV83, JAMS89] give some explanations on the method. The Simulated Annealing algorithm is detailed in algorithm 6. The functions used by this algorithm are:

- The *random_initialization* function consists in creating the initial partition by assigning randomly to each vertex the part in which it goes;
- $TEMPFACTOR$ is the cooling ratio to decrease the temperature;
- $MINPERCENT$ is used for testing whether the annealing run should be frozen;

Algorithm 6 Simulated Annealing

let $partition = random\_initialization$;
let $temperature = starting\_temperature$;
let $frozen\_counter = 0$;
while $frozen\_counter < 5$ **do**
  let $nb\_of\_accepted\_moves = 0$;
  while $loop\_nb < SEARCHSIZE$ **do**
    let $new\_partition = find\_partition\_neighbor(partition)$;
    if $energy(new\_partition) < energy(partition)$ **or**
    $\exp(\frac{energy(partition)-energy(new\_partition)}{temperature}) > rand(1.)$ **then**
      $nb\_of\_accepted\_moves = nb\_of\_accepted\_moves + 1$;
      $frozen\_counter = 0$;
      $partition = new\_partition$;
    $loop\_nb = loop\_nb + 1$;
  $temperature = TEMPFACTOR * temperature$;
  if $nb\_of\_accepted\_moves < MINPERCENT * SEARCHSIZE$ **then**
    $frozen\_counter = frozen\_counter + 1$;

- $SEARCHSIZE = SIZEFACTOR * |V|$ is the number of neighbors to find before a stage is raised. $|V|$ is the cardinal of the set of vertices V, and $SIZEFACTOR$ is a fixed value;
- The function $rand(n)$ returns a random number between 0 and n;
- The energy function (*i.e.* objective function) used is:

$$Ncut_\alpha(partition) = imb_\alpha(partition) + Ncut(partition)$$

  with

$$imb_\alpha(partition) = \alpha * \left(\max_{i \in \{1,...k\}} W(V_i) - \min_{i \in \{1,...k\}} W(V_i) \right)$$

  α is a parameter of the algorithm. It is useful to control the imbalance of the weight of the partition;
- The function $find\_partition\_neighbor$ selects randomly a vertex $v \in V$, and assigns v in a new random part different from the oldest.

6 Results and comparisons

The methods presented before are applied to resolve a practical problem which arises from the European Air Traffic Control. This problem is presented in subsection 6.1. A short view of the libraries used for making some comparisons between methods is given in subsection 6.2. Then algorithms applied to this Air Traffic Control problem are compared in subsection 6.3.

6.1 An Air Traffic Control problem: FABOP

The study was done at the LOG[3] laboratory on an air traffic control problem, called the Functional Airspace Block Optimized Process[4]. This problem is a graph partitioning problem. More precisely, this problem is a new sort a graph partitioning problem, the relaxed k-way graph partitioning problem. Let us describe this air traffic control problem.

"The primary purpose of the air traffic control system is to prevent a collision between aircraft operating in the system and to organize and expedite the flow of traffic" (Federal Aviation Administration [FAA97]). The first objective of air traffic control is safety, the second is efficiency. The FABOP project is a study in the "strategic" level of air traffic control, *i.e.*. our methods are applied long before any tactical control of aircraft. However, we are working for safety and efficiency. Our study only focuses on states of high air traffic in Europe, which are: Germany, France, United Kingdom, Switzerland, Belgium, Netherlands, Austria, Spain, Denmark, Luxembourg and Italy [BA05].

Each air traffic controller supervises a limited space, called an air traffic sector. Controllers have qualifications to work only on a set of sectors. These sets are called functional airspace blocks. The FABOP project consists in cutting the European airspace into blocks. Currently blocks almost never cross countries borders. A new organization of blocks only based on flows of aircraft and not on borders is studied. Because "it is well known that controller-controller coordination is easier and more effective inside an air traffic control unit (a block) than between air traffic control units" [Hal05], we search to maximize flows of aircraft inside blocks and further to minimize flows of aircraft between blocks.

The airspace, composed of blocks and sectors, can be viewed as a graph. The graph vertices are air traffic sectors and edges are flows of aircraft between sectors. The weight of a vertex is the number of aircraft which are going through the corresponding sector in a day. And the weight of an edge is the number of aircraft which are going from one sector of the edge to another. The weight of a vertex can be different than the sum of the weights of its connected edges, because arrival and departure of aircraft. Arrivals and departures could not be considered as a sector, because its size would be too large. Some aircraft are going out of the states of the study area. These aircraft are also counted in sectors weights. Thus, our air traffic problem is a partitioning problem with k, the number of functional airspace blocks into which we want to cut the airspace.

A recommendation of the European organization for the safety of air navigation, Eurocontrol, is to create functional airspace blocks of around 25 sectors [Eur06]. This number of sectors corresponds to the W_{opt} value defined in section 2. A second recommendation is to create blocks with a maximum of 50 sectors. Then, because there are 640 sectors in the study area, approximately 26 blocks have to be defined. Furthermore, the imbalance of the number of sectors per part must be fewer than: $\frac{50}{W_{opt}} = 2$ with $W_{opt} = \lceil \frac{640}{26} \rceil = 25$. Indeed, the imbalance of weight is more important than the imbalance of the number of sectors. Because, there is no recommendation from Eurocontrol about the weight (*i.e.* number of aircraft) of the blocks, we assume that the imbalance of the weight must not be larger than the imbalance of the number of vertices. Thus, the imbalance of the weight must be fewer than 2.

Regarding to the objectives: maximizing flows of aircraft inside blocks and mini-

[3] Laboratoire d'Optimisation Globale, www.recherce.enac.fr
[4] FABOP, www.recherche.enac.fr/opti/FABOP

mizing flows of aircraft between blocks, the appropriate objective function to use is the *Ncut* objective function.

To summarize, this air traffic problem is a relaxed k-way graph partitioning problem with $k \approx 26$, $v_{imb} \leq 2$ and $w_{imb} \leq 2$.

6.2 Libraries used and algorithms parameters settings

Four libraries and one implementation of a classical metaheuristic were used to make some comparisons with the Fusion Fission method.

6.2.1 The METIS library

We used the METIS library of George Karypis and Vipin Kumar [KK98a]:
glaros.dtc.umn.edu/gkhome/views/metis

The METIS library is the state of the art well known library used to make some comparisons between k-way partitioning problems.This library is based on the Multilevel method. It provides two graph partitioning applications, `pmetis` and `kmetis`. Their objective is to partition a graph into k equal-size parts while minimizing the *Cut* objective function. `pmetis` uses a recursive bisection, and `kmetis` uses a k-way algorithm.

This library is not directly designed for our problem, because it minimizes the *Cut* objective function and not the *Ncut* objective function and moreover because the imbalance of the partition is not an input of the algorithm.

6.2.2 The GRACLUS library

The GRACLUS library of Inderjit Dhillon, Yuqiang Guan and Brian Kullis [DGK04] is based on the METIS library and a kernel k-means algorithm. It can be found at: www.cs.utexas.edu/users/dml/Software/graclus.html

The `graclus` application can be set to minimize the *Ncut* objective function. Thus, it is a good adaptation of the METIS library to our problem. To obtain results as good as possible, we set the number of local search step to 4, while default is 0. We choose 4, because for our observations, there is no improvement of the quality of the partition above this number.

6.2.3 The JOSTLE library

The serial JOSTLE library of Chris Walshaw [Wal02] is another state of the art partitioning package: staffweb.cms.gre.ac.uk/ c.walshaw/jostle

The `jostle` program searches to minimize the *Cut* objective function. It tries to create a perfectly loaded balance, but has a customizable balance tolerance. This tolerance range is from 1.0 to 1.5. Thus, like METIS, the JOSTLE library is not perfectly designed for our problem. To find the lowest *Ncut* value of the graph with the `jostle` program, the 51 possible partitions of the air traffic control graph were computed, with `jostle` parameter *imbalance* $\in \{0,\ldots,50\}$, corresponding to *imbalance* $\in \{1.00,\ldots,1.50\}$. Results of this computations can be shown figure 2. The *Ncut* results are the results average of one hundred permutation of the same graph. The partition found with the lowest *Ncut* value is for the imbalance parameter set to 37.

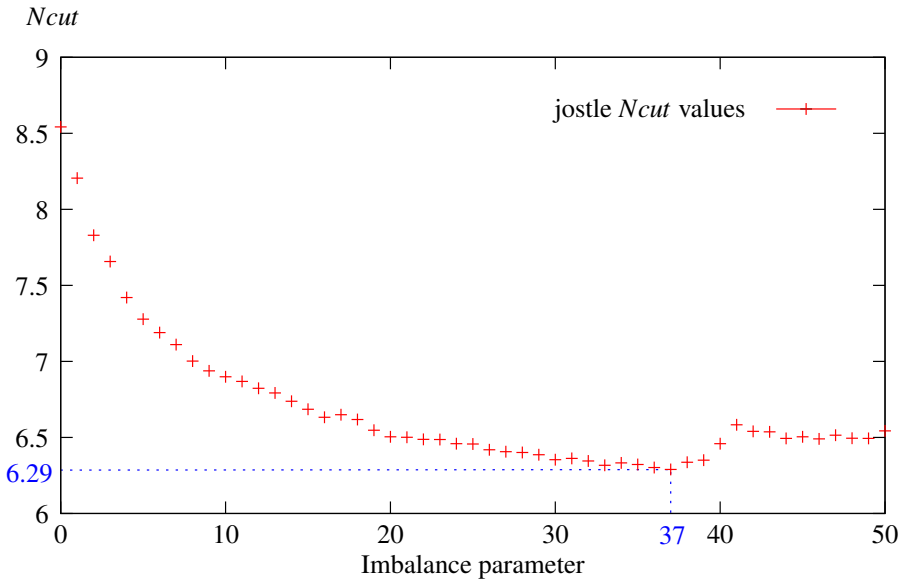

Figure 2: The 51 *Ncut* values return by jostle depending on its *imbalance* parameter, for the same graph

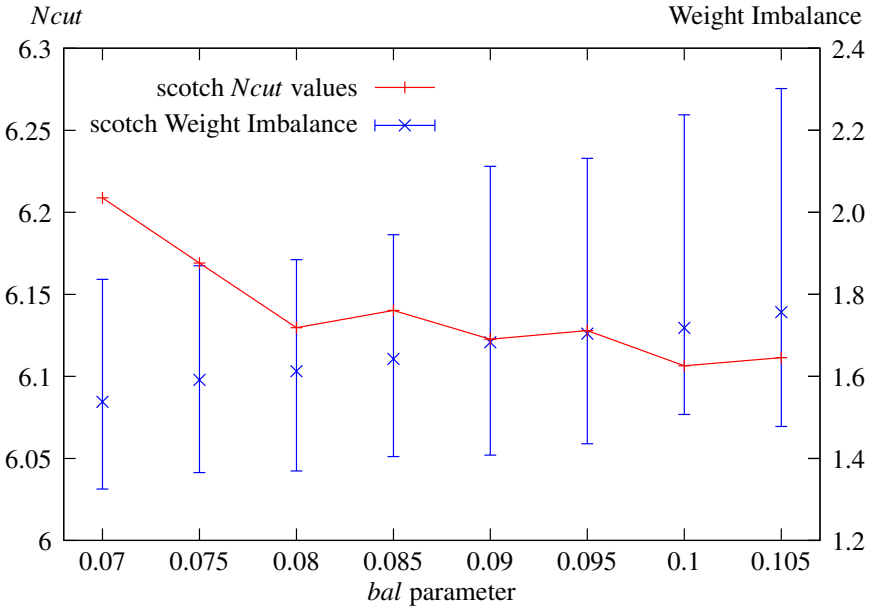

Figure 3: scotch *Ncut* results for different bal parameter's values

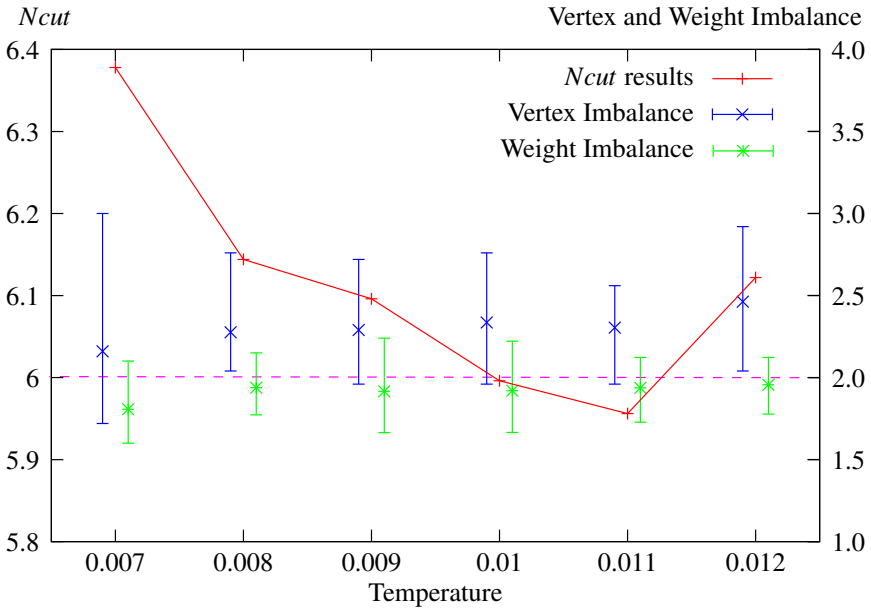

Figure 4: Simulated Annealing *Ncut* results for different *temperature* parameter's values. $\alpha = 16 * 10^{-6}$.

6.2.4 The SCOTCH library

We used the SCOTCH library of Franois Pellegrini [Pel06]:
www.labri.fr/perso/pelegrin/scotch

This is a high customizable library. The partitioning program is based on a dual bipartitioning mapping algorithm, which uses a Multilevel algorithm as a bipartitioning strategy. Partitions can be refined with different methods: Fiduccia-Mattheyses, Gibbs-Poole-Stockmeyer or greedy graph growing methods. The algorithm searches to decrease the *Cut* objective function, with a customizable imbalance of the weight of the parts.

It occurs that the defaults parameters give good results. The selection operator was not used in the strategy of the dual recursive bipartitioning. The imbalance ratio of the Fiduccia-Mattheyeses method was changed, which is originally set for a perfectly balanced partition. Figure 3 presents the average *Ncut* results of one hundred permutation of the graph for different values of the *bal* imbalance parameter. This parameter is limited up to 0.105 because upper values have weight imbalance which can go 20% up to the fixed limit of weight imbalance $w_{imb} = 2$. The mapping strategy strings used are: `b{job=t,map=t,poli=S,strat=MULTI}` with `MULTI=m{asc=FM,low=h{pass=10}FM,type=h,vert=80,rat=0.7}` and `FM=f{move=80,pass=-1,bal=0.1}`, where `bal=0.1` which corresponds to the lowest *Ncut* result of figure 3.

6.2.5 Simulated Annealing algorithm

An adaptation of the Simulated Annealing algorithm presented by David Johnson, Cecilia Aragon, Lyle McGeoch and Catherine Schevon in [JAMS89] is used, as describe in section 5.2.

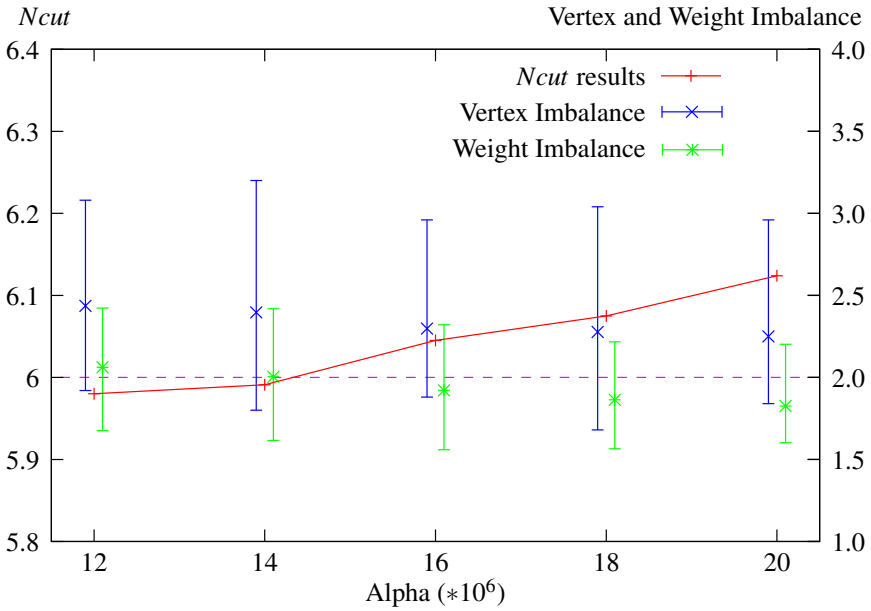

Figure 5: Simulated Annealing *Ncut* results for different *alpha* parameter's values.

Because Multilevel algorithms are extremely fast, less than one second of computation, the computation time of the Simulated Annealing algorithm has to be limited. We decide to allow about 2 minutes of computations to the algorithm. It is not a to long time, but allows relatively good results. Thus, to obtain this computation time, we fix $TEMPFACTOR = 0.95$ and $SIZEFACTOR \in \{14, 15, 16\}$ depending on the starting temperature. In their article, Johnson *et all* dissuade to spend too much time at the end of the algorithm, thus we fix $MINPERCENT = 0.01$, which means that after 5 times a stage is consecutively raised with less than 1% of accepted moves, the process is frozen and then stop. Moreover, they advise to start the algorithm with a percentage of accepted moves of the first stage roughly equal to 0.4. This percentage of accepted moves corresponds to $starting\_temperature = 0.007$. Figure 4 displays the *Ncut* average and range results of 20 permutations of the air traffic control graph for different $starting\_temperature$ parameters. The best temperature value is $temperature = 0.011$. Figure 5 displays the *Ncut* average and range results of 100 permutations of the graph for different α parameters. . Because $\alpha = 12 * 10^{-6}$ and $\alpha = 14 * 10^{-6}$ have a weight imbalance upper than 2, the best available *Ncut* average corresponds to $\alpha = 16 * 10^{-6}$.

6.2.6 Fusion Fission algorithm

Because rules are highly customizable, it is hard to explain the choice made. Fusion between two parts tends to create a new big part. So it is interesting to eject some nucleons of this new part to create a smaller part. Fission of a part splits it in two smaller parts. Thus, it is only interesting to eject few connected nucleons, but not to decrease the size of the parts. Then, the rules are initialized with:

$$rule\_fusion = \{0.5; 0.2; 0.2; 0.07; 0.03; 0.01\}$$

$$rule\_fission = \{0.8; 0.15; 0.05; 0.0\}$$

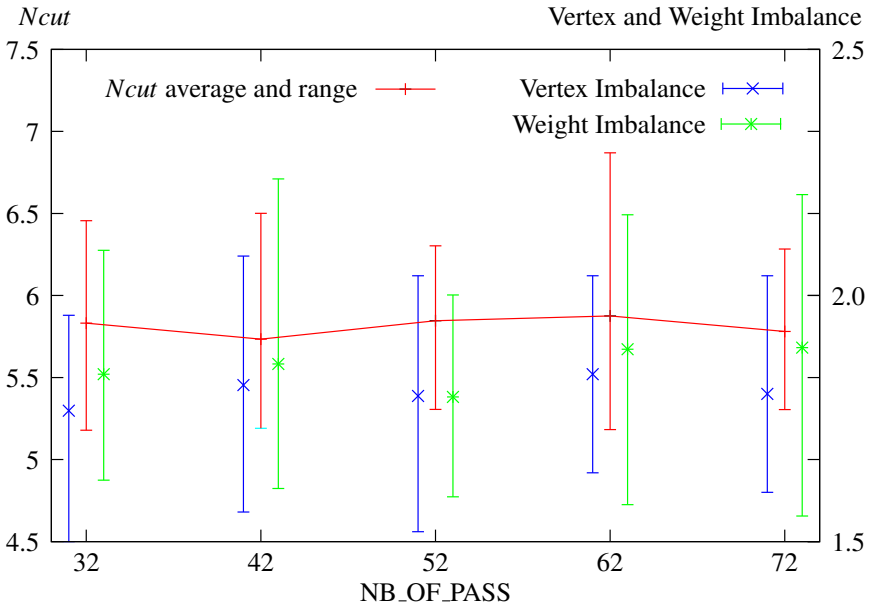

Figure 6: Fusion Fission *Ncut* results for different NB_OF_PASS parameter's values with CHAINED=1.6.

$$percent = \{0.0; 0.01; 0.02; 0.03; 0.04; 0.05\}$$

The other parameters are:

- The rules are updated with RULE_ADJUSTMENT = 0.001
- The maximal energy after the initialization is INIT = 14
- See figure 6 for the choice of the NB_OF_PASS parameter. This is the pass number of the second loop of the algorithm 2. The results presented in this figure are the average results of twenty permutations of the air traffic control graph. Results are close, however, the lowest *Ncut* average is raised for NB_OF_PASS = 42
- The function choice_fusion has for internal parameters: MULT = 0.2 and ADD = 0.3
- Figure 7 presents results obtained for different CHAINED values. The results are obtained for twenty permutations of the main graph. Each *Ncut* average is very close to the others. Vertex imbalance average and weight imbalance average are all under the threshold. Thus, the chained fission process control parameter is set to CHAINED = 1.7, because it is the lowest *Ncut* average.

6.3 Comparisons between methods

The air traffic control problem presented upper (subsection 6.1) gives us a weighted graph on vertices and edges. Such a graph can be made for different days of data, but in this paper only results which are obtain for the Friday, 17^{th} June 2005 air traffic

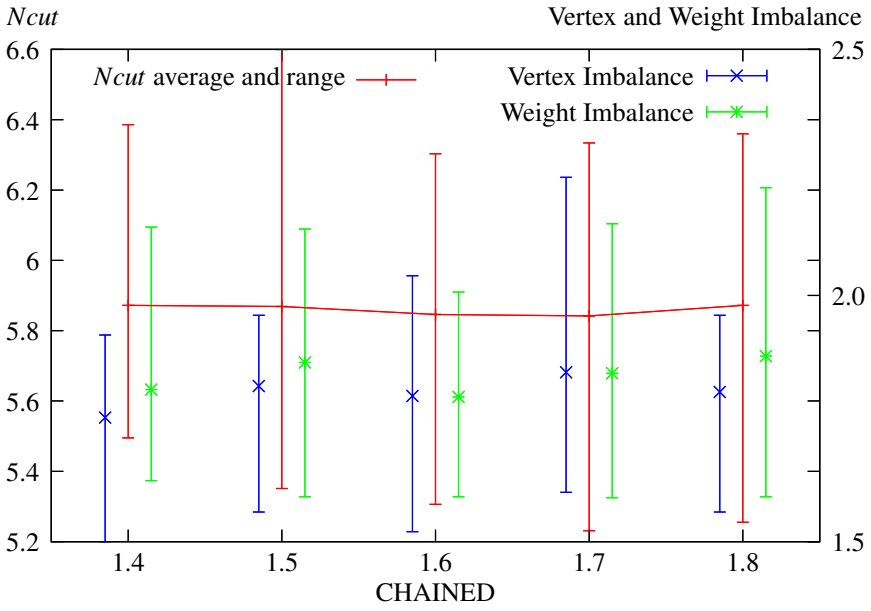

Figure 7: Fusion Fission *Ncut* results for different CHAINED parameter's values with NB_OF_PASS=42.

control data of Eurocontrol were presented. The graph is stored using the CHACO graph input file format describes in [HL95a]. We choose this format for compatibility reasons. The METIS, JOSTLE, SCOTCH or GRACLUS libraries use also this graph input file format.

During a graph partitioning process, a method can return a configuration of particularly low or high energy, depending on the input configuration of the graph. To avoid this kind of problem, benchmark were made with one hundred permutation of the initial graph, and then, average and range of the results were returned. A permutation only consists in renaming each vertex of the graph. Indeed, the structure of the graph is unchanged.

All results presented in this paper are found with an Intel Pentium 4, 3GHz processor with 1Go RAM computer, which uses a GNU/Linux Debian operating system. Every specific graph partitioning package partitions the graph in less than one second, but metaheuristics are greatly slower with an average of 121 seconds for Fusion Fission and 122 seconds for Simulated Annealing. The computational times of the metaheuristics was limited. They can find better solutions with more time. We have found that a roughly 2 minutes computation time is enough but not too long regarding results found.

The computational results are shown in table 1. Algorithms were sorted by their *Ncut* values. First, it can be noticed that metaheuristics have better results than the other algorithms. But the difference of computation time between metaheuristics and the other algorithms can explain these differences. The Fusion Fission algorithm has the best *Ncut* average. It is just followed by the Simulated Annealing algorithm with an *Ncut* average which is 3.2% worth than the Fusion Fission. Then come the Multilevel methods. The scotch algorithm is the first of them. Its *Ncut* average is 4.3% larger than for the Fusion Fission algorithm. Then comes the graclus algorithm. The three state of the art graph partitioning packages which are not design to solve this relaxed

Algorithms	FF	SA	scotch	graclus	jostle	pmetis	kmetis
Ncut	5.85	6.04	6.11	6.25	6.29	6.88	7.40
Vertex imbalance	1.80	2.30	1.80	1.81	1.82	1.88	1.75
Weight imbalance	1.87	1.92	1.72	1.75	1.37	1.13	1.03
Computation time	121s	122s	$\ll$1s	$\ll$1s	$\ll$1s	$\ll$1s	$\ll$1s
Distance from FF	-	3.2%	4.3%	6.4%	7.0%	15.0%	20.9%

Table 1: Average of the results of 100 permutations of the air traffic control graph

k-way graph partitioning problem come at the end. The jostle algorithm has a better *Ncut* average than the METIS package, because of its imbalance parameter.

Results found by the algorithms must have a vertex imbalance $v_{imb} \leq 2$ and a weight imbalance $w_{imb} \leq 2$ as it was explained in section 6.1. Each algorithm has an average vertex imbalance around 1.8, except the Simulated Annealing which is greatly out-range with a vertex imbalance of 2.30. For the four better algorithms, the weight imbalance is relatively high, but under the threshold $w_{imb} = 2$. The weight imbalance of the jostle algorithm has been chosen in section 6.2.3 to have the best *Ncut* value. Of course, the METIS library has the lowest weight imbalance because its imbalance can not be changed.

To summarize, the Fusion Fission algorithm finds the best results. The Simulated Annealing does not respect the vertex imbalance criteria, which means that its objective function must be changed. And the scotch algorithm finds the best solution of all Multilevel algorithms.

For each algorithm, *Ncut* values average and range of the one hundred permutation of the air traffic control graph can be shown by figure 8. For all algorithms, the jostle algorithm has the largest range of *Ncut* values. Its lower range is 5.68 and its upper range is 40% greater. However, the standard deviation of *Ncut* results of the jostle algorithm is 0.34. This is the same standard deviation for Fusion Fission, 0.33. But standard deviation of *Ncut* results are lower for the other algorithms: 0.26 for kmetis, 0.25 for Simulated Annealing, 0.23 for graclus, 0.18 for scotch and 0.12 for pmetis. This means that for the same graph, very different partitions qualities can be found with the jostle algorithm. This is also true for the Fusion Fission. But because Fusion Fission finds partitions of great quality, its baddest partitions are not worst than the baddest of Simulated Annealing or the baddest of jostle. The relatively high standard deviation found for each algorithm confirms the utility to apply the algorithms to one hundred permutations of the same graph.

The vertex and weight imbalance of the partitions found for each algorithm can be shown by figure 9. The horizontal line represents the threshold of $v_{imb} \leq 2$ or $w_{imb} \leq 2$. The Simulated Annealing vertex average is really above this threshold, and it seems that it has too much weight imbalance above this threshold too. As it was expected, the jostle, pmetis and kmetis algorithms have very low weight imbalance compared to the threshold. As we explain in subsection 6.2, this is due to their very strict weight partitioning constraints. However, both of their vertex imbalance have a too big range, with an upper range of approximately $2.5 \gg 2$. Partitions found by Fusion Fission, scotch and graclus have good vertex imbalance, both on average and range. The Fusion Fission algorithm most respects the relaxed k-way graph partitioning problem constraints for this air traffic control graph.

The Fusion Fission algorithm not only found partitions for a fixed k, but for a range of values around k (see section 4). Then, when the Fusion Fission algorithm resolves

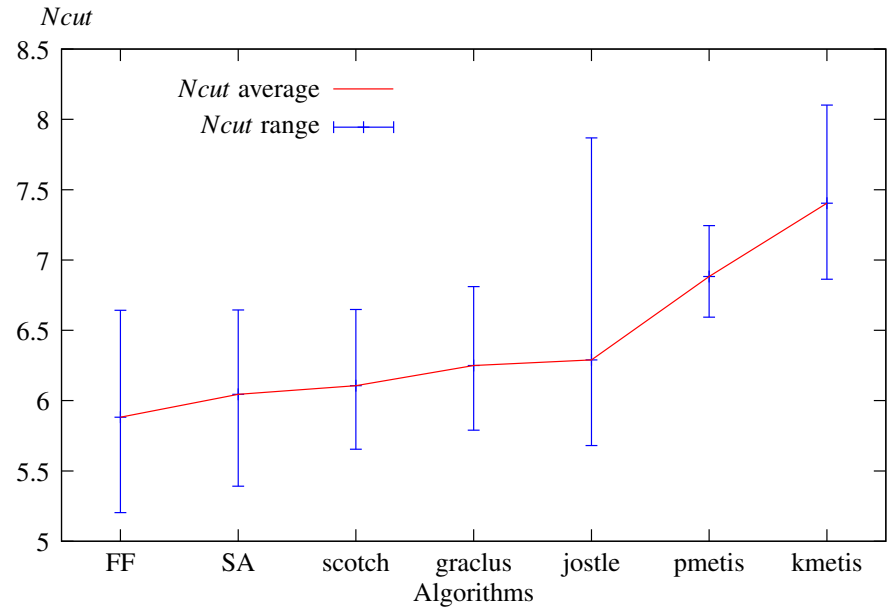

Figure 8: Algorithms *Ncut*'s average and range results of 100 permutations of the air traffic control graph

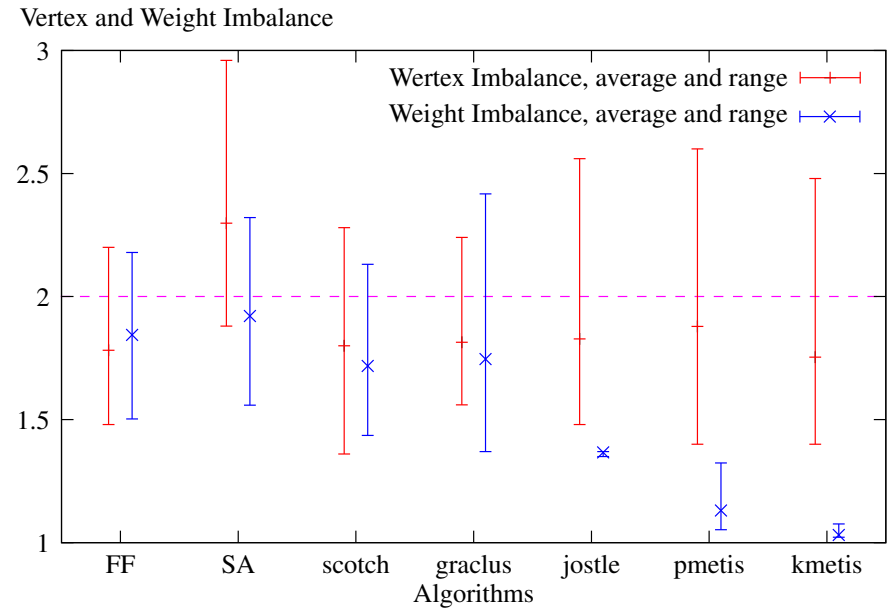

Figure 9: Algorithms average and range results of the vertex and weight imbalance of 100 permutations of the air traffic control graph

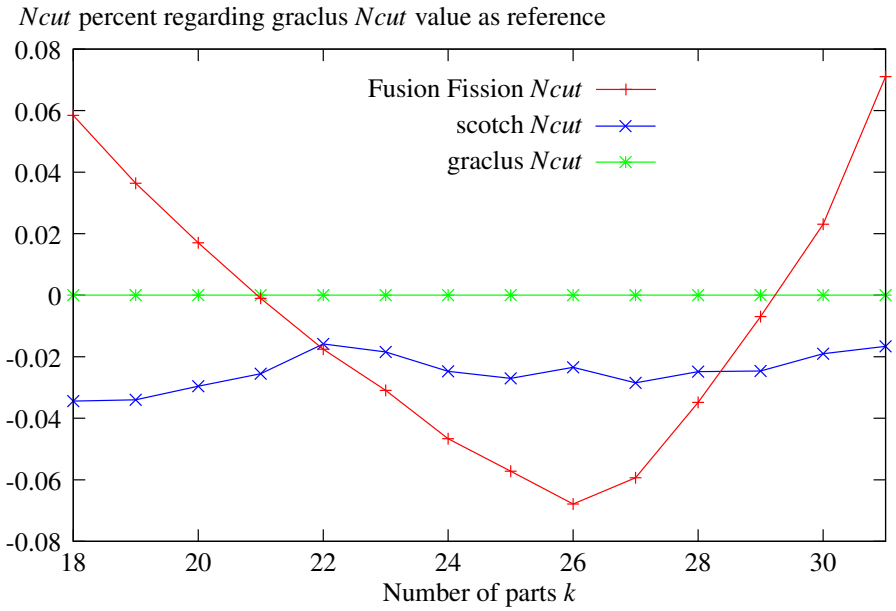

Figure 10: Comparisons between Fusion Fission results for $k = 26$, with GRACLUS and SCOTCH results for $k \in \{18, \ldots, 31\}$. Each algorithm results are presented as rate variation between the algorithm's *Ncut* average and the corresponding *Ncut* average of graclus

the air traffic control graph partitioning problem, it finds partitions with a partition's cardinal range: $\{19, \ldots 30\}$. And, for more than 90% of the graph permutations, a partition's cardinal range: $\{18, \ldots 31\}$. To compare all of the partitions found by the Fusion Fission algorithm with other partitions, the two better Multilevel algorithms, scotch and graclus, computed the one hundred permutations of the air traffic control graph for $k \in \{18, \ldots 31\}$. Results of the computation of the Fusion Fission algorithm with $k = 26$ and for the 13 computations of the scotch algorithm and for the 13 computations of the graclus algorithm, each with $k \in \{18, \ldots 31\}$, are displayed figure 10. In this figure, each algorithm results are presented as rate variation between the algorithm's *Ncut* average and the corresponding *Ncut* average of graclus. The partitions found by the Fusion Fission algorithm are better than those of scotch for $k \in \{22, \ldots, 28\}$. And the partitions found by the Fusion Fission algorithm outperforms the partitions found by graclus for $k \in \{21, \ldots, 29\}$.

7 Conclusion

In this paper, a new graph partitioning problem is presented, the relaxed k-way graph partitioning problem. It arises in an air traffic control problem which consists in cutting the European sky into parts. State-of-the-art graph partitioning packages were not designed to solve this problem. Thus a new method, the Fusion Fission has been created to solve this problem. However, it could be interesting to adapt this method to other graph partitioning problem.

There is some comparisons between Multilevel, Simulated Annealing and Fusion

Fission algorithms in this paper. Multilevel algorithms are very fast. The Fusion Fission algorithm is difficult to parameterize, also is Simulated Annealing. But the best Multilevel algorithm, the jostle algorithm, is also difficult to parameterize. Regarding the results, the Fusion Fission algorithm finds the better partitions, then comes Simulated Annealing and after Multilevel algorithms.

The Fusion Fission method has proved that it is a powerful method to partition graphs. It outperforms the Simulated Annealing algorithm and Multilevel algorithms. We hope that our Fusion Fission approach to the relaxed k-way graph partitioning problem will inspire further work to other graph partitioning problem and to other combinatorial problems.

Our future research will try to simplify the Fusion Fission method. We will also applied it to different partitioning subjects, such as image segmentation and k-way graph partitioning.

References

[AHK97] Charles J. Alpert, Jen-Hsin Huang, and Andrew B. Kahng. Multilevel circuit partitioning. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 530–533, 1997. [CiteSeer url] .

[BA05] Charles-Edmond Bichot and Jean-Marc Alliot. A theoretical approach to defining the European Core Area . Technical report, École Nationale de l'Aviation Civile / Centre d'Étude de la Navigation Arienne, France, 2005.

[DGK04] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kullis. Kernel k-means, Spectral Clustering, and Normalized Cuts . In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining*, pages 551–556, 2004. [CiteSeer url] .

[DMP95] R. Diekmann, B. Monien, and R. Preis. Using helpful sets to improve graph bisections. In *Proceedings of the DIMACS Workshop on Interconnection Networks and Mapping and Scheduling Parallel Computations*, pages 57–73, 1995. [CiteSeer url] .

[Eur06] Eurocontrol. The impact of fragmentation in European ATM/CNS . Technical report, Eurocontrol, 2006.

[FAA97] FAA. *Air Traffic ControL : FAA Order 7110.65K* . Federal Aviation Administration (U.S. Department of Transportation), 1997.

[FM82] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[GJS76] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[Gre01] William A. Greene. Genetic algorithms for partitioning sets. *International Journal on Artificial Intelligence Tools*, 10(1-2):225–241, 2001.

[Hal05] Anders Hallgren. Restructuring european airspace: functional airspace blocks. *Skyway*, pages 20–22, autumn 2005.

[HL95a] Bruce Hendrickson and Robert Leland. *The Chaco User's Guide* . Sandia National Laboratories, 2.0 edition, 1995.

[HL95b] Bruce Hendrickson and Robert W. Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of Supercomputing*, 1995. [CiteSeer url] .

[JAMS89] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. Optimization by simulated annealing: an experimental evaluation; part i, graph partitioning. *Operations Research Society of America*, 37(6):865–892, 1989.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. [CiteSeer url] .

[KK98a] George Karypis and Vipin Kumar. *METIS : A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices* . University of Minnesota, 4.0 edition, september 20 1998.

[KK98b] George Karypis and Vipin Kumar. Multilevel Algorithms for Multi-Constraint Graph Partitioning . In *Proceedings of Supercomputing*, pages 1–13, 1998. [CiteSeer url] .

[KL70] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970.

[KLS97] Pascale Kuntz, Paul Layzell, and Dominique Snyers. A colony of ant-like agents for partitioning in vlsi technology. In *Proceedings of the European Conference on Artificial Life*, pages 417–424, 1997.

[LG99] A. E. Langham and P. W. Grant. A Multilevel k-way Partitioning Algorithm for Finite Element Meshes using Competing Ant Colonies . In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, volume 2, pages 1602–1608, 1999.

[Pel06] Franois Pellegrini. *Schotch and LibScotch* . ENSEIRB - LaBRI, Universit de Bordeaux I, 4.0 edition, january 31 2006.

[SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. [CiteSeer url] .

[SWC04] A.J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph-Partitioning . *Journal of Global Optimization*, 29:225–241, 2004.

[TB91] E. G. Talbi and P. Bessiere. A parallel genetic algorithm for the graph partitioning problem. In *Proceedings of the ACM International Conference on Supercomputing*, pages 312–320, 1991.

[Wal02] Chris Walshaw. *The serial JOSTLE library user guide* . University of Greenwich, 3.0 edition, july 8 2002.

Table 2: The European airspace partition and the best Fusion Fission partition

Partitions	Number of part	$Ncut$	w_{imb}	v_{imb}
The 6/17/2005 partition	55	25.10	4.64	3.33
The best FF partition	26	4.87	1.69	1.64

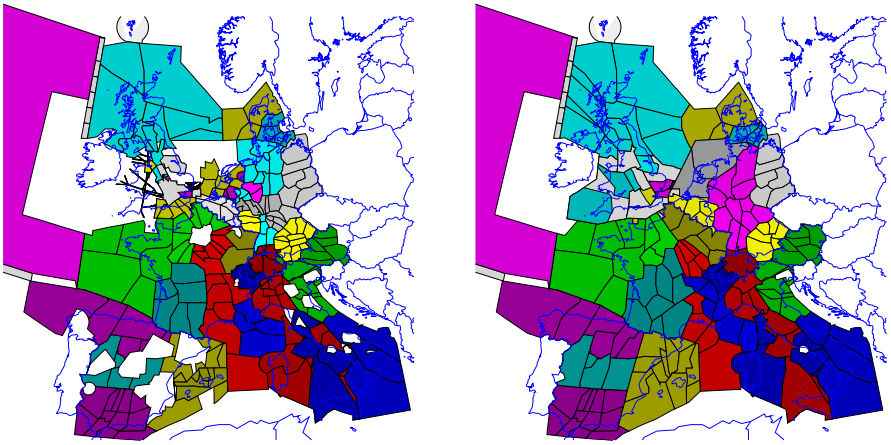

Figure 11: The real European airspace partitioning at flight levels 180 and 280

[Wal04] Chris Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems . *Annals of Operations Research*, 131:325–372, 2004.

[WCM00] C. Walshaw, M. Cross, and K. McManus. Multiphase mesh partitioning . *Applied Mathematical Modeling*, 25:123–140, 2000. [CiteSeer url] .

A European airspace partitioning

The European airspace area presented in this paper is actually partitioned into 55 blocks. Table 2 presents a comparison between the European airspace partition of June, 17^{th} 2005 and the best partition found by the Fusion Fission algorithm into 26 parts. Because the numbers of parts of the two partitions are very different, the $Ncut$ comparison is not relevant. The imbalance of the vertices number of the current European airspace partition into controls centers is really above the Eurocontrol recommendation. The European airspace partition of June, 17^{th} 2005 at flight level[5] 180 and 280 can be shown figure 11. The best Fusion Fission partition found is presented figure 12, it takes four hours of computation.

[5] A flight level is the altitude of an aircraft in foot unit divided by one hundred

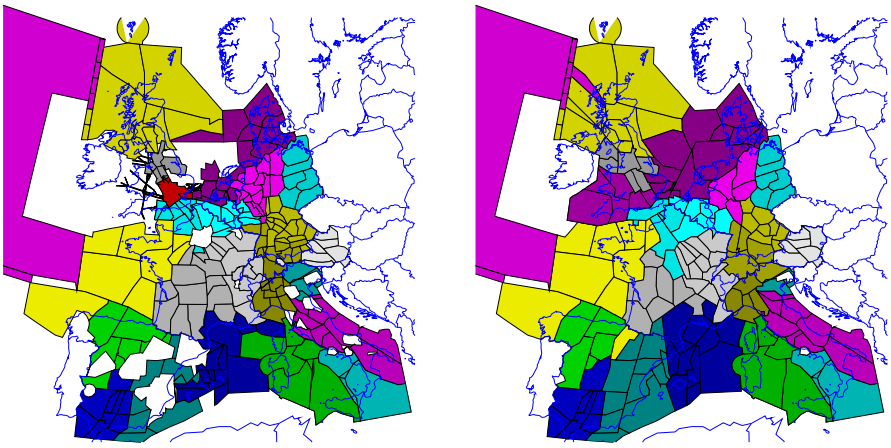

Figure 12: The best European airspace partition found with the Fusion Fission algorithm at flight levels 180 and 280