Decomposition Tehniques for Solving Frequency Assigment Problems (FAP) *A Top-Down Approach*

Lamia Sadeg-Belkacem[1,2,3], Zineb Habbas[2], Fatima Benbouzid-Si Tayeb[1] and Daniel Singer[2]
[1]*LCMS, ESI, Algiers, Algeria*
[2] *LCOMS, University of Lorraine, Ile du Saulcy, 57045 Metz cedex, France*
[3]*Laboratory of Applied Mathematics, Military Polytechnic School, Algiers, Algeria*

Keywords: Frequency Assignment Problem, Constraint Satisfaction Problem, Graph Clustering, Genetic Algorithm.

Abstract: This paper deals with solving *MI-FAP* problem. Because of the NP-hardness of the problem, it is difficult to cope with real FAP instances with exact or even with heuristic methods. This paper aims at solving MI-FAP using a decomposition approach and mainly proposes a *generic Top-Down approach*. The key idea behind the generic aspect of our approach is to link the decomposition and the resolution steps. More precisely, two generic algorithms called *Top-Down* and *Iterative Top-Down* algorithms are proposed. To validate this approach two decomposition techniques and one efficient *Adaptive Genetic Algorithm* (AGA-MI-FAP) are proposed. The first results demonstrate good trade-off between the quality of solutions and the execution time.

1 INTRODUCTION AND RELATED WORKS

The increasing development of new wireless services has led to foster studies on Frequency Assignment Problem (FAP). FAP was proved to be *NP-hard* (Hale, 1980) and more details on FAP can be found in (Aardal et al., 2003). The present work deals with the Minimum Interference Frequency Assignment Problem (*MI-FAP*) that aims to allocate a reduced number of frequencies to transmitters/receivers while minimizing the overall set of interferences in the network. Because of the NP-hardness of the problem it is very difficult to cope with real instances with both exact or heuristic algorithms. Although several exact approaches have been proposed (enumerative search, B&B, ...), they are not efficient when dealing with realistic instances. In order to address large instances of FAP, numerous heuristics and metaheuristics have been proposed. One can cite (Maniezzo and Carbonaro, 2000) who applied an Ant Colony Optimization metaheuristic to MI-FAP. (Kolen, 2007) proposed a Genetic Algorithm but it is very time consuming. (Voudouris and Tsang, 1995) examined the application of the Guided Local Search to FAP. However, all those metaheuristics have not confirmed their performances on large instances.

In the last decade, some works have investigated decomposition techniques in order to address large instances of FAP proposing to exploit structural properties of the problem. (Koster et al., 1998) and (Allouche et al., 2010) used *Tree Decomposition* to decompose the problem and used exact algorithms for its resolution. This approach improved several lower bounds for hard instances of CALMA (CALMA-website, 1995). (Colombo and Allen, 2007) proposed a generic algorithm for decomposing the problem into a collection of sub-problems connected by a *cut* and solving them in a recursive way by metaheuristics. More recently, (Fontaine et al., 2013) developed a local search algorithm guided by a tree decomposition.

This paper presents the first investigations towards a generic method based on decomposition combined with metaheuristics for solving large optimization problems. The MI-FAP problem is used as a particularly representative and interesting target application. The objective here is twofold, to solve the problem near optimally and in the shortest possible time. The generic method leads to an original *Top-Down approach* solving first the sub-problem associated with the cut and the sub-problems associated with the clusters afterwards. Two versions of the method are proposed. The first one called *Top-Down* is a backtrack-free algorithm and the second one is an improved version called *Iterative-Top-Down*. To validate our propositions two decomposition algorithms are defined, based both on the well known Min-Cut decomposition algorithm of (Stoer and Wagner, 1997). One is called *Balanced Min-Cut Weigthed Decomposition* and the other *Balanced Min-Cut Cardinality Decomposition*. A robust and fast Genetic Algorithm for MI-FAP has also been developed in order to solve the sub-problems. For the refinement of the global solution, the 1-opt local search was used. One can notice that this generic method can be used with any other decomposition, any other resolution algorithm. The quality of solutions and the runtime of the different approaches, with and without decomposition, are compared on benchmarks given in CALMA project. The results indicate that our best strategy can significantly improve the computation time without any significant loss of quality of the solution.

The rest of this paper is organized as follows. Section 2 gives a formal presentation of FAP. Section 3 presents a generic Top-Down approach for the resolution of FAP involving a decomposition step. Two variants of Balanced Min-Cut Decompositions are presented in Section 4. Section 5 presents AGA_MI-FAP and 1-opt local search method. Section 6 presents the first results of this approach while Section 7 concludes the paper.

2 FORMULATIONS OF FAP

2.1 Partial Constraint Satisfaction Problems (PCSP)

Definition 1. Constraint Satisfaction Problem. *A Constraint Satisfaction Problem (CSP) is defined as a triple $P =< X,D,C >$ where*

- $X = \{x_1,...,x_n\}$ *is a finite set of n variables.*
- $D = \{D_1,...,D_n\}$ *is a set of n finite domains. Each variable x_i takes its value in the domain D_i.*
- $C = \{C_1,...,C_m\}$ *is a set of m constraints. Each constraint C_i is defined on an ordered set of variables $S_i \subseteq X$ called the scope of C_i.*
  For each constraint C_i a relation R_i specifies the authorized values for the variables defined in S_i. This relation R_i can be defined intentionally as a formula or extentionally as a set of tuples, $R_i \subseteq \prod_{x_k \in S_i)} D_k$ (subset of the cartesian product).

Definition 2. Constraint Graph. *A binary CSP $P =< X,D,C >$ can be represented by a Constraint Graph $G =< V,E >$ where $V = X$ and $E = \{(x_i x_j) : (x_i x_j) \in C\}$.*

Definition 3. PCSP (Koster et al., 1998). *A binary Partial Constraint Satisfaction Problem (PCSP) is defined a a quintuple $P =< X,D,C,P,Q >$ where $< X,D,C >$ is a binary CSP as defined previously. P is a set of constraint -penalty functions $P = \{P_{(x_i x_j)} : D_i \times D_j \to \mathbf{R}\}$ where $(x_i x_j) \in C$ and Q is a set of variable-penalty functions $Q = \{Q_{x_i} : D_i \to \mathbf{R}\}$ where $x_i \in X$.*
Each value taken by a variable $x_i \in X$ can be subject to a penalty. Moreover, a constraint $(x_i x_j) \in C$ indicates that some combinations of values for x_i and x_j are also penalized. The objective when solving a PCSP is to select a value for each variable $x_i \in X$ such that the total penalty is minimized.
A solution of a PCSP is represented by a complete assignment of values to each variable $x_i \in X$ denoted $< d_1, d_2, \ldots, d_n >$ where $d_i \in D(_i)$. The cost of a solution is defined as the sum of all constraint-penalties and variable-penalties, as follows:

$$\sum_{x_i \in X} Q_{x_i}(d_i) + \sum_{(x_i x_j) \in C} P_{(x_i x_j)}(\{d_i, d_j\})$$

Solving a PCSP consists in finding a solution with a minimum cost.

Remark 1. *Among the set of constraints, those that must not be violated are called "hard" constraints while the others are "soft" constraints. C_h and C_s will denote the sets of hard and soft constraints respectively.*
A PCSP is sometimes called weighted CSP and denoted as a quintuple $P =< G,D,C,P,Q >$ where $G =< V,E >$ is the constraint graph associated with the CSP $< X,D,C >$. In this way, it can naturally be viewed as a weighted constraint graph.

2.2 Modeling MI-FAP as a PCSP

Many variants of FAP belong to the class of PCSP.

Definition 4. *A MI-FAP is a PCSP defined by $< T,F,C,P,Q >$ where $T = \{t_1, t_2, \ldots, t_n\}$ is a set of transmitters, $F = \{F_1, F_2, \ldots, F_n\}$ with F_i the set of possible frequencies that can be assigned to the transmitter t_i and $C = \{C_1, C_2, \ldots, C_m\}$ is a set of binary constraints. A constraint between two transmitters t_i and t_j indicates that communication from t_i may interfere with communication from t_j. An interference occurs in general when the difference between the frequencies assigned to the transmitters is less than a given threshold. For each constraint C_k in C with scope $S_k = \{t_i, t_j\}$, some* *combinations of values $(f_i, f_j) \in F_i \times F_j$ are penalized (constraint-penalty). Moreover, some transmitter can be associated with preassigned frequencies. For such transmitter all the other possible frequencies are penalized (variable-penalty).*

A solution to a MI-FAP is represented by a complete assignment of frequencies $< f_1, f_2, \ldots, f_n >$ to each transmitter $t_i \in T$, with $f_i \in F_i$. Solving a MI-FAP consists in finding a solution minimizing:

$$\sum_{t_i \in T} Q_{t_i}(f_i) + \sum_{(t_i t_j) \in C} P_{(t_i t_j)}(\{f_i, f_j\})$$

3 SOLVING MI-FAP WITH DECOMPOSITION

3.1 Motivation

In this paper, an original ”Top-Down” approach is investigated for FAP, where, the decomposition and solving steps are closely related. Two generic algorithms are presented. The first one called *Top-Down* is a backtrack-free and fast algorithm. The second algorithm is an iterative version of the first one.

3.2 Top-Down Algorithm

Given a MI-FAP problem represented as a weighted graph, the *Top-Down* algorithm (Algorithm 1) decomposes first (in Line 1) the problem into a collection of *k* sub-problems (clusters). Each edge (i,j) between a pair of clusters C_k and C_l is a constraint between two antennas *i* and *j* of the MI-FAP problem. The variables associated with *i* and *j* are called *boundary variables*. The set of all the *boundary variables* constitutes the resulting *cut*. Then (in Line 2), it solves the cut sub-problem and gives rise to one partial solution ”*Sol_cut*” and its cost ”*Cost_cut*”. In Lines[3-5], the *k* sub-problems are solved in sequential or in parallel. Notice that this step considers that all *boundary variables* are already instantiated, and this significantly reduces the size of the resulting clusters. The global solution ”*Sol*” and its cost ”*Cost*” are computed in Lines 6-8 respectively. The final, optional step improves the quality of the solution (Line 9).

3.3 Iterative Top-Down Algorithm

Once Algorithm 1 has computed a solution for the cut problem, all the *boundary variables* are instantiated. This reduces the search space associated with the clusters. To avoid this drawback, an improved

Algorithm 1: Top-Down algorithm.

```
Input : $G =< V,E >$
Ouptut: A global solution $Sol$ and its $Cost$
1: Decompose(G, C1, C2, ... Ck, cut)
2: Solve(cut, Sol_cut, Cost_cut)
3: for i = 1 to k [1] do
4:     Solve(Ci, Sol_Ci, cost_Ci, Sol_cut )
5: end for
6: $Sol \leftarrow Sol\_C1 \odot^2 Sol\_C2, \ldots \odot^2 Sol\_Ci$
7: $Cost\_Clusters \leftarrow \sum_{i=1}^{k} Cost\_Ci$
8: $Cost \leftarrow Cost\_Clusters + Cost\_cut$
9: Improve (Sol, Cost)
```

version of the former algorithm called *Iterative Top-Down* algorithm (see Algorithm 2) is proposed. It relaxes the cut sub-problem by cancelling the instantiation of some boundary variables of the cut.

Algorithm 2: Iterative Top-Down algorithm.

```
Input : $G =< V,E >$,
$Max\_Iter$: nb. max. of iterations, H : heuristic
Ouptut: A global solution $Sol$ and its $Cost$
1: Decompose(G, C1, C2, ... Ck, cut)
2: Init_sol(G, Sol, Cost) // finds one initial solution
3: Sol_cut $\leftarrow Sol[cut]$
4: for i = 1 to Max_Iter do
5:     Release(G,cut,H,Sol,Sol_cut,Sol_cut',Cost_cut')
6:     for i = 1 to k [1] do
7:         Solve(Ci, sol_Ci, cost_Ci, Sol_cut')
8:     end for
9:     $Current\_Sol \leftarrow sol\_C1 \odot^2 sol\_C2, \ldots \odot^2 sol\_Ck$
10:    $Cost\_Clusters \leftarrow \sum_{i=1}^{k} cost\_Ci$
11:    $Current\_Cost \leftarrow Cost\_Clusters + Cost\_cut'$
12:    Improve ($Current\_Sol, Current\_Cost$)
13:    if $Current\_Cost < Cost$ then
14:        $Sol \leftarrow Current\_Sol$
15:        $Cost \leftarrow Current\_Cost$
16:    end if
17: end for
```

Algorithm 2 proceeds as follows: given a FAP instance modelled as a weighted graph $G =< V,E >$, the procedure ”Decompose”(Line 1) gives a partition of G into k clusters $C1, \ldots Ck$. The procedure ”Init_sol” finds an initial solution Sol of the problem by using either a random strategy or a heuristic method. Sol_cut is the solution of the cut problem obtained by projecting Sol on the cut, according to line 3. To increase the search space of the sub-problems, the procedure ”Release” is called in Line 5. This procedure removes the instantiation of some *boundary variables*. The choice of these *boundary variables* to be restored further depends on a given heuristic H. This procedure returns a partial solution Sol_cut' of the cut. All the sub-problems are then solved, as in the previous algorithm. The current solution $Current\_Sol$ and its cost $Current\_Cost$ are given in lines 9 and 10. This current solution is improved (Line 12). For more diversification and to escape from local minima the above process is repeated a certain number of times

[1]It can be a sequential or a parallel loop.

[2]$\odot$ is the concatenation of two partial solutions.

(Max_Iter). To validate the iterative Top-Down algorithm, the choice of the *boundary variables* to be removed from the cut is done by **MIC heuristic**: given a cluster *Ck*, the *boundary variable* to be removed from the cut is the variable *i* with the Maximum Internal Cost (MIC). The Internal Cost of *i* in *Ck* is: $\sum_{(i,j)\in E;\ j\in Ck} w_{ij}$.

4 DECOMPOSITIONS METHODS

In this section two algorithms are proposed for decomposing MI-FAP. They are both based on a well known algorithm due to Stoër (Stoer and Wagner, 1997) for the Min-Cut problem of a weighted graph with a Minimum Cut in terms of weight. To generalize this idea to the *k*-partitioning problem, a divisive algorithm is used in a recursive way. Moreover, while the original algorithm does not exploit the size of the clusters, a “balanced decomposition” which is of particular interest for parallel solving is targeted in this work. Finally, as the resolution algorithms are closely linked to the decomposition, several form of decomposition are investigated. In particular, two algorithms are presented: BMCWD and BMCCD.

4.1 Preliminary: Min-Cut Algorithm

Before detailing the approach, the key notion behind the Min-Cut algorithm due to Stoër that is the Minimum *s-t* cut (see Theorem 1) are described.

Theorem 1. (Min-Cut of a graph (Stoer and Wagner, 1997)): *Let s and t be two vertices of a graph G. Let $G(s/t)$ be the graph obtained by forcing the vertices s,t to be in two different clusters and let $G/\{s,t\}$ be the graph obtained by merging s and t. Then a Minimum Cut of G can be obtained by taking the minimum of the Minimum cut of $G(s/t)$ and a Minimum cut of $G/\{s,t\}$.*

Intuitively, this theorem means that either there exists a Min-cut of *G* that separates *s* and *t* and then the Minimum *s-t* cut of G is a Min-Cut of *G*, or there is none and so the Min-Cut of $G/\{s,t\}$ fits. The algorithm saves the Minimum *s-t* cut for arbitrary $s,t \in V$, and merges them to find a Min-Cut in the graph. The Min-Cut is the minimum of the $|V|-1$ cuts found. The main loop in Algorithm 3 calls the Min-cut-step procedure Algorithm 4 to split the current graph *G* into two clusters C_{cur1} and C_{cur2} connected by a Min-Cut with a weight called w_{Cur}.

The Min-cut-step procedure (Algorithm 4) adds to a given set *A* initialized to *s*, the **M**ost **T**ightly **C**onnected **V**ertex with *A* (**MTCV(A)**) until *A* equals

Algorithm 3: Min-Cut.[3]

```
Input : $G = <V, E>$
Output: $C_1, C_2, w$
1: $C_1 \leftarrow \emptyset$ ; $C_2 \leftarrow \emptyset$
2: $s \leftarrow$ elementof(V)    /* $s$ randomly selected */
3: $w \leftarrow w_G$
4: while $|V| > 1$ do
5:     Min-cut-step($G, s, v_{end-1}, v_{end}$)
6:     $C_{cur1} = V - \{v_{end}\}$ ; $C_{cur2} = \{v_{end}\}$
7:     $w_{cur} \leftarrow Cut(C_{cur1}, C_{cur2})$
8:     $G \leftarrow Shrink(G, v_{end-1}, v_{end})$
       /* $v_{end-1}$, $v_{end}$ : the two vertices returned by Min-cut-step */
9:     if $w_{cur} < w$ then
10:        $w \leftarrow w_{cur}$
11:        $C_1 \leftarrow C_{cur1}$ ; $C_2 \leftarrow C_{cur2}$
12:    end if
13: end while
```

V. The added vertices $v_{end}, v_{end-1} \in A$ will compose the current clusters in Algorithm 3. The cut of these clusters is proven to be Minimum v_{end}-v_{end-1}-*cut* of the initial graph *G* (in (Stoer and Wagner, 1997)).

As a consequence, v_{end}, v_{end-1} are merged (Shrink) in the rest of the algorithm This operation is repeated until $|V| = 2$. The Min-cut of the initial graph *G* is then the minimum of the $|V-1|$ cuts found. The starting node *s* can be the same for the whole algorithm or it can be selected arbitrarily in each computation phase as well.

Algorithm 4: Min-cut-step.

```
Input : $G = <V, E>$, $a \in V$
Output: $v_{end-1}, v_{end} \in V$
1: $A \leftarrow \{a\}$
2: while $A \neq V$ do
3:     $A \leftarrow A \cup MTCV(A)$
4: end while
```

Proposition 1. *The theoretical complexity of the Min-cut decomposition is in $O(|V||E| + |V|^2 log|V|)$.*

The proof is given in (Stoer and Wagner, 1997)

4.2 Decomposition Algorithms for FAP

A great number of different approaches have been proposed in the literature for a graph decomposition. None of them is much better or worse than the other in all cases. In fact, the quality of a decomposition is closely related to the nature of the problem to be solved. In this work, the goal assigned to this decomposition step is to allow solving large-scale size problems in a reasonable time, while obtaining near optimal solutions. To approximate an optimal solution, one possibility is to minimize the cost of the cut with respect to the global cost of the graph. This is the first expected property for the proposed decomposition. Therefore, a second property for our decomposition is to produce ”balanced clusters” which can

[3] $w_G = \sum_{(ij)\in E} w(ij)$, Cut(X, Y) = $\sum_{(ij)\in E, i\in X, j\in Y} w(ij)$.

be solved independently. The two properties lead to what will be called an “efficient decomposition”.

Definition 5. **(“Efficient decomposition”)** *Let $G =< V,E >$ be a graph decomposed into a partition P of k clusters C1, C2 . . . Ck.*

- *%cut is the ratio $\frac{w}{w_G}$ where w is the the weight of the cut and w_G is the weight of G.*
- *b is defined as the parameter to measure the balance of a decomposition as follows: $b = \frac{Min(|C1|,|C2|,...,|Ck|)}{Max(|C1|,|C2|,...,|Ck|)}$, where $|Ci|$ is the number of vertices in Ci.*

An “efficient decomposition” is a decomposition with b close to 1 and %cut close to 0.

4.2.1 BMCWD Algorithm

BMCWD algorithm (*Balanced Min-Cut Weighted Decomposition*) described by algorithm 5 aims at searching for a well balanced and an efficient decomposition with a small cut even if it is not Minimum.

Algorithm 5: BMCWD.

```
Input : $G =< V,E > Max_{iter}$, w_threshold, b_threshold
Output: $C_1, C_2, w_{balance}$
1:  $C_1 \leftarrow \emptyset$ ; $C_2 \leftarrow \emptyset$
2:  $s \leftarrow$ elementof($V$)   /*randomly selected */
3:  $w_{balance} \leftarrow w_G$
4:  while $|V| > 1$ do
5:      $k \leftarrow 0$ ; $flag \leftarrow 0$ ; $T \leftarrow \emptyset$
6:      while ($flag = 0$ and $k < Max_{iter}$) do
7:          $k \leftarrow k+1$
8:          Min-cut-step($G, s, v_{end-1}, v_{end}$)
9:          $C_{cur\_1} \leftarrow V - \{v_{end}\}$ ; $C_{cur\_2} \leftarrow \{v_{end}\}$ ; $w_{cur} \leftarrow Cut(C_{cur\_1}, C_{cur\_2})$
10:           /* property 1 of “efficient decomposition” */
11:           if $w_{cur} > w\_threshold$ then
12:               $flag \leftarrow 1$
13:           else if $k <= Max_{iter}$ then
14:               save ($T, [v_{end}, v_{end-1}, s, w_{cur}]$)
15:               $s \leftarrow$ elementof($V$)   /* a new initial vertex randomly selected */
16:           else
17:               /* select from T [$v_{(end)i}, v_{(end-1)i}, a_i$] with max $w_{(cur)i}$ */
18:               select_max($T, v_{(end)i}, v_{(end-1)i}, s_i, w_{(cur)i}$)
19:               $s \leftarrow s_i$
20:               $v_{end-1} \leftarrow v_{(end-1)i}$ ; $v_{end} \leftarrow v_{(end)i}$ ; $w_{cur} \leftarrow w_{(cur)i}$
21:           end if
22:       end while
23:       $G \leftarrow Shrink(G, v_{end-1}, v_{end})$ /* $v_{end}, v_{end-1}$: the 2 last vertices put in $A$ */
24:       /* property 2 of “well balancing */
25:       $b = \frac{Min(|C_{cur\_1}|,|C_{cur\_2}|)}{Max(|C_{cur\_1}|,|C_{cur\_2}|)}$   /* the balance condition */
26:       if $b > b\_threshold$ then
27:           if $w_{cur} < w_{balance}$ then
28:               $w_{balance} \leftarrow w_{cur}$ ; $C_1 \leftarrow C_{cur\_1}$ ; $C_2 \leftarrow C_{cur\_2}$
29:           end if
30:       end if
31: end while
```

In order to improve the cut, the BMCWD Algorithm merges the two last added nodes of the Cut(v_{end} and v_{end-1}) only if the value of the cut separating v_{end} and v_{end-1} denoted (w_{cur}) is large enough. Otherwise the previous values are saved, and the execution of the algorithm is aborted without calling the Shrink function. A new Min-Cut step is then executed with a new initial vertex. In that case, the value of w_threshold corresponds to $\frac{w_G}{|E|}$ where $|E|$ is the number of edges in the current graph G. This step is executed a number of times equal to Max_{iter}, after which the most connected vertices v_{end} and v_{end-1} are merged.

Proposition 2. *The theoretical complexity of BMCWD algorithm is $O(k_{max} \times (|V||E| + |V|^2 log|V|))$ where k_{max} is the maximum number of iterations.*

4.2.2 BMCCD Algorithm

The BMCCD (*Balanced Min-Cut Cardinality Decomposition*) algorithm is a variant of BMCWD which minimizes the number of edges in the cut. A simple way to link the two algorithms is to consider that all the edges have a weight equal to 1 (in original graph). In that case, the BMCCD and th BMCWD algorithms are equivalent.

5 SOLVING METHOD

5.1 Genetic Algorithm: Informal Presentation and Useful Notations

This section presents a Genetic Algorithm (GA) dedicated to MI-FAP resolution corresponding to the function **solve** in Algorithms 1, 2 of section 3.

In standard GA crossover and mutation probabilities are predetermined and fixed. Consequently, the population becomes premature and falls in local convergence early. To avoid this drawback an Adapative Genetic Algorithm (AGA) is proposed. The following notations are introduced to facilitate the presentation of AGA algorithm:

Let $\mathcal{P} =< X,D,C,P,Q >$ be a PCSP and $G =< V,E >$ its weighted graph ($V = X$, $E = C$ and $|V| = n$).

- $N[v_i] = \{v_j \in V | (v_i, v_j) \in E\}$ is the Neighbourhood of the vertex v_i in G.
- $s = (f_1, f_2, \ldots, f_n)$ denotes a solution of $\mathcal{P}$ where $f_i \in D_i \; \forall i \in \{1,\ldots,n\}$.
- Fitness (v_i,s) = $\sum_{v_j \in N[v_i], (v_i,v_j)\, unsat} w(f_i, f_j)$ is the cost associated with v_i for solution s.
- Fitness (s) = $\frac{1}{2} \sum_{i=1}^{n} Fitness(v_i, s)$ is the cost associated with solution s.

5.2 Presentation of AGA-MI-FAP

In this study, the MI-FAP problem is represented as a weighted graph $G =< V,E >$. A chromosome is a set of $|V|$ genomes, where each genome corresponds to the frequency f_i assigned to the vertex $v_i \in V$. In other words a chromosome represents a possible solution to the MI-FAP problem.

An initial population is defined and three operations (selection, mutation, crossover) are performed to generate the next generation. This procedure is repeated until a convergence criterion is reached. The sketch of AGA-MI-FAP is given by Algorithm 6.

Algorithm 6: AGA-MI-FAP.

```
Input: $p_{m0}$, $p_{c0}$: initial mutation and crossover probabilities, $\Delta p_m$, $\Delta p_c$: mutation and
crossover probabilities rates.
1:  $p \leftarrow$ Initial_Population;
2:  if local mimima then
3:      $p_m = p_m - \Delta p_m$    ;    $p_c = p_c + \Delta p_c$
4:  else
5:      $p_m = p_{m0}$    ;    $p_c = p_{c0}$
6:  end if
7:  old_p = p
8:  repeat
9:      for all parent_i chromosome in old_p, i is the ith chromosome do
10:             in parallel
11:             parent_j = a selected chromosome in old_p using the tournament algorithm
12:             if $p_c$_ok then
13:                 offspring_i ← Crossover(parent_i, parent_j), where offspring_i will be
                  the ith chromosome in a future population.
14:             else
15:                 offspring_i = parent_i
16:             end if
17:             if $p_m$_ok then
18:                 offspring_i = Mutation(offspring_i)
19:             end if
20:         end for
21:     until convergence
```

Algorithm 7: Crossover(p_1, p_2).

```
1:  Fitness_new[$p_1$]= Fitness[$p_1$]
2:  Fitness_new[$p_2$]= Fitness[$p_2$]
3:  for all i = 1 to n  do
4:      Fitness_new[$p_1$](i)=Fitness_new[$p_1$](i)+ $\sum_{v_j \in N[v_i]} Fitness[p_1](j)$
5:      Fitness_new[$p_2$](i)=Fitness_new[$p_2$](i)+ $\sum_{v_j \in N[v_i]} Fitness[p_2](j)$
6:  end for
7:  Temp = Fitness_new[$p_1$] - Fitness_new[$p_2$]
8:  Let j = k such that Temp[k] is the largest element in Temp.
9:  for all i = 1 to n  do
10:     $offspring[i] = \begin{cases} p_1[i] & \text{if } i \neq j \text{ and } v_i \notin N[v_j] \\ p_2[i] & \text{otherwise} \end{cases}$
11: end for
```

The performance of AGA-MI-FAP is tightly dependent on crossover and mutation operators. The mutation operator is used to replace the values of a certain number of genomes, randomly chosen in the parent population, in order to improve the fitness of the resulting chromosome. The mutation occurs with a probability p_m, named mutation probability. The crossover operator is used to improve the fitness of a part of the chromosome (Algorithm 7). Crossover appears with a probability p_c called the crossover probability. p_m and p_c are two complementary parameters which have to be fine tuned. A good value for p_c avoids the local optima (diversification) while p_m enables the GA to improve the quality of solutions (intensification). In the proposed AGA both parameters are dynamically modified to reach a good balance between the intensification and the diversification.

Since all chromosomes of a given population are independent, crossover and mutation operations can be processed concurrently. A classical GA has been first implemented and tested. The AGA algorithm has been then tested. The results demonstrate that the AGA-MI-FAP significantly improves the quality of the solution as compared with the classical GA.

6 EXPERIMENTAL RESULTS

6.1 Environment Considerations and Description of Benchmarks

All implementations have been developped using C++. The tests have been performed on the supercomputer Romeo[1]. Only one single 8-core processor at 2.4 Ghz was used in this experimentation.

We tested our approach on real-life instances of CALMA-project (CALMA-website, 1995). The set of instances consists in two parts. The CELAR instances are real-life problems from a military application. The GRAPH instances are randomly generated problems. We only use the so-called MI-FAP instances (Table 1). In this paper we are only concerned by instances 6, 7 and 8 of CELLAR and instances 5, 6, 11 and 13 of GRAPH. Other instances were not considered because they are easy.

Table 1: Benchmark characteristics.

Instance	Original graph		Reduced graph		Best cost
	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	
CELAR06	200	1322	100	350	3389
CELAR07	400	2865	200	816	343592
CELAR08	916	5744	458	1655	262
GRAPH05	200	1134	100	416	221
GRAPH06	400	2170	200	843	4123
GRAPH11	680	3757	340	1425	3080
GRAPH13	916	5273	458	1877	10110

6.2 BMCWD vs. BMCCD

This section presents comparative results obtained for BMCWD and BMCCD algorithms by using 2 and 3 clusters. Notice that the instance CELAR08 is excluded from this test and all tests concerning approaches based on decomposition because it is already decomposed in several and unbalanced clusters. To compare these algorithms we based on the measure

[1]https://romeo1.univ-reims.fr/, University of Champagne-Ardenne

parameters %_*cut* previously defined, and %_*Bn* corresponding to the ratio of *boundary nodes* resulting from the decomposition. This last parameter plays an important role in our tests because our approach is mainly focused on the number of *boundary nodes*. The balance threshold parameter is fixed to 0.8.

Table 2: % of cut and *boundary nodes* found by BMCWD and BMCCD with 2 and 3 clusters.

Instance	Method	k=2		k=3	
		%_cut	%_Bn	%_cut	%_Bn
CELAR06	BMCWD	**2.8**	30.5	**4**	43
	BMCCD	**3.33**	17	4.39	22.5
CELAR07	BMCWD	**0.002**	36.5	**0.7**	47.25
	BMCCD	**0.98**	5.75	**1.12**	7.25
GRAPH05	BMCWD	**6**	54.5	7.1	75
	BMCCD	7.85	47	12	61.5
GRAPH06	BMCWD	7.5	53.5	8.9	61.5
	BMCCD	7.37	46.25	11.43	74.5
GRAPH11	BMCWD	**5**	61.76	8.5	75.29
	BMCCD	8.86	51.91	11.71	63.38
GRAPH13	BMCWD	7.8	67.36	9.5	69.65
	BMCCD	8.76	50.54	12.33	66.37

Table 2 shows that for 2 clusters the parameter %_*cut* is low, while for 3 clusters this parameter increases. This is due to the hierarchical nature of our decomposition algorithm. The number of *boundary nodes* varies from instance to another for both algorithms but BMCCD produces less *boundary nodes* in general.

6.3 AGA-MI-FAP Alone

In this section, we report the best costs (cost) and average costs (avg_cost) obtained by using AGA-MI-FAP alone (50 executions). Initial Mutation and Crossover probabilities are fixed experimentally to 1 and 0.2 respectively, $\Delta p_m = \Delta p_c = 0.1$, and the population size is fixed to 100.

Table 3 shows clearly the efficiency of AGA-MI-FAP algorithm. Indeed, optimal solutions have been obtained for the majority of instances and near-optimal solutions are obtained on the rest.

Table 3: Performance of AGA-MI-FAP.

Instance	cost(avg_cost)	cpu(s)	Best cost
CELAR06	**3389(3389)**	37	3389
CELAR07	343691(343794)	312	343592
CELAR08	**262(264)**	571	262
GRAPH05	**221(221)**	35	221
GRAPH06	**4124(4128)**	222	4123
GRAPH11	3119(3191)	2246	3088
GRAPH13	10392(10812)	3700	10110

6.4 Approaches using Decomposition

In this section we present the results for Top-Down and Iterative Top-Down on 2 clusters and 3 clusters.

6.4.1 Top-Down Algorithm

Table 4: Results of Top-Down with 2 and 3 clusters.

Instance	Method	k=2		k=3	
	BMC	cost(avg_cost)	cpu	cost(avg_cost)	cpu
CELAR 06	WD	3389(3822)	13	3422(5302)	10
	CD	3389(3499)	15	3402(4922)	10
CELAR 07	WD	363897(434297)	161	353800(2455320)	74
	CD	343592(1374412)	120	343912(1385114)	84
GRAPH 05	WD	221(382)	26	267(869)	15
	CD	221(236)	21	257(844)	14
GRAPH 06	WD	4126(4431)	160	5019(9090)	63
	CD	4126(4311)	175	4193(7107)	83
GRAPH 11	WD	3466(4038)	763	8779(19983)	310
	CD	3256(4060)	677	7616(12367)	231
GRAPH 13	WD	11015(14256)	1690	25140(32571)	690
	CD	10796(12511)	1810	22333(31035)	575

Table 5: Results of Iterative Top-Down with 2 and 3 clusters.

Instance	Method	k=2		k=3	
	BMC	cost(avg_cost)	cpu	cost(avg_cost)	cpu
CELAR 06	WD	3401(3873)	20	3420(3820)	19
	CD	3423(3861)	23	3401(4067)	20
CELAR 07	WD	425218(1476655)	219	424126(1353999)	153
	CD	343810(536322)	179	343691(444117)	164
GRAPH 05	WD	255(391)	21	238(1897)	18
	CD	221(345)	25	225(643)	20
GRAPH 06	WD	4340(7810)	169	4861(8084)	151
	CD	4298(5631)	147	4632(6569)	161
GRAPH 11	WD	4119(7500)	751	4127(9751)	551
	CD	4195(7182)	892	3673(7490)	632
GRAPH 13	WD	17183(19495)	1998	19846(26701)	1456
	CD	13278(18757)	1969	17255(25885)	1501

Table 5 reports the results obtained with Top-Down algorithm on 2 and 3 clusters. The quality of solutions of Top-Down algorithm decreases when the number of clusters increases. This is due to the increasing ratio of boundary nodes leading to a search space reduction . We also observe that generally the results obtained using Top-Down algorithm based on BMCCD algorithm are better than those obtained by using BMCWD. This is due to the same observation made about the parameter %_*cut* .

6.4.2 Iterative Top-Down Algorithm

Table 4 reports results obtained by Iterative Top-Down algorithm for 2 and 3 clusters. The populations size considered is 30 and number of iterations is fixed to 10. In general larger the population is better is the solution. The results obtained by Iterative Top-Down algorithm are encouraging and clearly improves the simple Top-Down algorithm (k=3). In general, Iterative algorithm maintains its performance even when the number of clusters increases, while the execution time decreases. The performance of the simple Top-Down algorithm decreases considerably with increasing number of clusters while the Iterative Top-Down one is much more stable.

6.5 Direct vs. Decomposition

Table 6 summarizes the results obtained by direct algorithm and best results obtained by approaches via decomposition. The parameter σ_cpu is the speed-up defined as $\frac{|CPU1|}{CPU2}$, where CPU1 and CPU2 are the ex-

Table 6: Comparing direct and decomposition approaches.

Instance	Direct		Via decomposition		σ_*cost*(%)	σ_*cpu*
	cost	cpu	cost	cpu		
CELAR06	3389	37	3389	**13**	**0.00**	**2.84**
CELAR07	343691	312	343592	120	**-0.02**	**2.60**
GRAPH05	221	35	**221**	**21**	**0.00**	1.66
GRAPH06	4124	222	4126	**160**	0.04	1.38
GRAPH11	3119	1946	3256	677	4.39	2.87
GRAPH13	10392	3700	10796	1810	3.88	2.04

ecution times of direct approach and that via decomposition respectively . The row σ_*cost* shows clearly that the results are comparable with the quality of the solutions on all instances. However, the row σ_*cpu* outlines clearly the benefit of approaches via decomposition in term of cpu time. this corresponds to our first objective aiming to solve large problems in short time near to optimality.

Table 7: Comparison with recent decomposition algorithms.

Instance	Our approach		All 10		Fon 13	
	cost	cpu(s)	cost	cpu(s)	cost	cpu(s)
CELAR06	**3389**	**13**	3389	212	3389	93
CELAR07	**343592**	**120**	343592	607	343592	317
GRAPH05	**221**	**21**	-	-	221	10
GRAPH06	4126	**160**	-	-	4123	240
GRAPH11	3256	677	-	-	3080	2762
GRAPH13	10796	1810	-	-	10110	3196

6.6 Comparison with Related Works

Table 7 compare the best results we obtained by our algorithms based on decomposition and the best results of (Allouche et al., 2010) and (Fontaine et al., 2013)) which both exploit Tree Decomposition of problems to be solved.

Our results are comparable to those presented in (Fontaine et al., 2013) in terms of quality of the solution but are better in terms of CPU-time.

7 CONCLUSIONS

In this paper, a Top-Down approach is developed for solving hard instances of MI-FAP problem near to optimality in short time.

To validate experimentally this approach:

- Two decomposition methods based on a Min-Cut algorithm were implemented. The first one called BMCWD aims to minimize the global weight of the cut. The second one called BMCCD aims to minimize the number of edges of the cut.
- An adaptive genetic algorithm (AGA-MI-FAP) was proposed to solve the initial problem without decomposition or for solving the sub-problems.
- The 1-opt local search heuristic was used to improve the global solution.

The quality of the solutions and the runtime of the different approaches, with and without decomposition, were compared on instances of CALMA project. Almost instances were solved using AGA-MI-FAP. When solving decomposed MI-FAP, optimal or near-optimal solutions were obtained in a short time with the proposed method. The Iterative Top-Down algorithm have good performances even when the number of clusters increases. This promising result leads to investigate further this decomposition approach. The first results obtained in this work indicate that the best strategy proposed can significantly improve the computation time without any significant loss of quality of the solution.

Several perspectives to this work will be investigated: different decomposition methods and criteria, other exact or heuristic algorithms to solve the clusters.

REFERENCES

Aardal, K., Van Hoessel, S., Koster, A., Mannino, C., and Sassano, A. (2003). Models and solution techniques for frequency assignment problems. *4OR, Quaterly Journal of the Belgian, French and Italian Operations Research Sciences*, 1:261–317.

Allouche, D., Givry, S., and Schiex, T. (2010). Towards parallel non serial dynamic programming for solving hard weighted csp. In *Proc. CP'2010*, pages 53–60.

CALMA-website (1995). *Euclid Calma project*. ftp://ftp.win.tue.nl/pub/techreports/CALMA/.

Colombo, G. and Allen, S. M. (2007). Problem decomposition for minimum interference frequency assignment. In *Proc. of the IEEE Congress in and Evolutionary Computation, Singapor*.

Fontaine, M., Loudni, M., and Boizumault, S. (2013). Exploiting tree decomposition for guiding neighborhoods exploration for vns. *RAIRO-Operations Research*, 47/2:91–123.

Hale, W. K. (1980). Frequency assignment: Theory and applications. 68/12:1497–1514.

Kolen, A. (2007). A genetic algorithm for the partial binary constraint satisfaction problem: an application to a frequency assignment problem. *Statistica Neerlandica*, 61/1:4–15.

Koster, A., Van Hoessel, S., and Kolen, A. (1998). The partial constraint satisfaction problems: Facets and lifting theorem. *O. R. Letters*, 23(3-5):89–97.

Maniezzo, V. and Carbonaro, A. (2000). An ants heuristic for the frequency assignment problem. *Computer and Information Science*, 16:259–288.

Stoer, M. and Wagner, F. (1997). A simple min-cut algorithm. *Journal of the ACM*, 44/4:585–591.

Voudouris, C. and Tsang, E. (1995). Partial constraint satisfaction problems and guided local search. Technical report, Department of Computer Science,University of Essex. Technical Report CSM-25.