Solving the Frequency Assignment Problem with Polarization by Local Search and Tabu

Philippe Galinier †‡, Michel Gendreau †¶,
Patrick Soriano †§ and Serge Bisaillon †

†Centre de recherche sur les transports, Université de Montréal,
C.P. 6128, succursale Centre-ville, Montréal, Québec, Canada H3C 3J7

‡Département de génie informatique, École Polytechnique de Montréal,
C.P. 6079, succursale Centre-ville, Montréal, Québec, Canada H3C 3A7

‡Département d'informatique et recherche opérationnelle, Université de Montréal,
C.P. 6128, succursale Centre-ville, Montréal, Québec, Canada H3C 3J7

§Service de méthodes quantitatives, École des Hautes Études Commerciales,
3000, chemin de la Côte-Sainte-Catherine, Montréal, Québec, Canada H3T 2A7

1 Introduction

Wireless communication networks have undergone a dramatic expansion over the past two decades. Given the continuing fast growth in demand for wireless services, Frequency Assignment Problems (FAP) play a key role in the planning of such networks [12]. Several variants of FAP have been studied over the past ten years. A common feature of these problems is the presence of distance or spacing constraints imposed on pairs of frequencies in order to avoid (or at least reduce) interferences between geographically close communication equipments. Problems arising in radio-mobile networks (i.e. cellular telephone networks, personal communications systems) generally comprise only distance constraints while problems arising in radio-link networks have other additional constraints. Their formal expression is therefore more complex. In 1993, the CELAR (the French "Centre d'Électronique de l'Armement") built a series of simplified versions of Radio Link Frequency Assignment Problems (RLFAP) based on data taken from a real network [3]. The Frequency Assignment Problem with Polarization (FAPP) that we study here is an extension of this latter problem. It was the topic of the 2001 International Challenge organized by the French operations research society ROADEF in collaboration with CELAR (see [4, 9] for more details). Strictly speaking, the FAPP no longer is a frequency assignment problem since polarities in addition to frequencies need to be assigned, which makes the expression of the distance constraints (also referred to as *electro-magnetic constraints*) more complex.

Over the last few years, various solution techniques for tackling frequency assignment problems have been proposed in the literature (see [5, 8, 13]). The vast majority of these focus on the mobile radio setting and the most successful applications for realistic sized settings are based on the use of meta-heuristics and in particular of tabu search.

In this paper, we consider an approach based on local search and tabu search [1, 7] to solve the FAPP. Our procedure includes some original features, including a specialized neighborhood, heuristics to determine critical variables and values, a diversification technique, an auto-adaptive mechanism to set the tabu list and a pre-processing based on consistency techniques inherited from constraint programming.

The remainder of this paper is organized as follows. The general principle of our local search approach is presented in section 2. Section 3 focuses on solution techniques. Computational experiments and results are then presented and analyzed in section 4. Finally, some concluding remarks and further perspectives are proposed in section 5.

2 A local search approach for solving the FAPP

2.1 Problem definition

In the FAPP, each link is associated given polarity and frequency domains. A potential solution or configuration is any assignment of both a polarity and a frequency to each link. The FAPP is a constrained optimization problem: the constraints of the problem are the so-called imperative constraints (IC), while the electro-magnetic constraints (EMC) are used to define the objective. Each EMC can be satisfied according to 11 degrees ranging from 0 (representing a perfect way to satisfy the constraint) to 10.

Considering any potential solution S, $g(S)$ will denote the number of imperative constraints violated in S and $f_k(S)$ $(0 \leq k \leq 10)$ the number of EMC violated in S with respect to degree k. To simplify the presentation, we introduce a fictitious additional degree of feasibility noted $k = 11$ for which any ECM is always satisfied,*i.e.* $f_{11}(S) = 0, \forall S$. We will say that a solution S is k-feasible if $g(S) = 0$ and $f_k(S) = 0$. We also define the degree of feasibility $\mathcal{R}(S)$ of a configuration S as the smallest k for which S is k-feasible: if $g(S) > 0$ then $\mathcal{R}(S) = \infty$, else $\mathcal{R}(S)$ is the smallest integer $k, 0 \leq k \leq 11$ such that $f_k(S) = 0$.

2.2 Search strategy

To solve the FAPP, we perform two successive phases. First, the *feasibility phase* consists in finding a solution with a minimum degree of feasibility k. Then, the *optimization phase* consists in finding a k-feasible solution that minimizes f_{k-1} and secondarily $\sum_{j \leq k-2} f_j$. Let us consider the three following subproblems:

- R_k : to find S such that $g(S) = 0$ and $f_k(S) = 0$
- O_k : to minimize $f_{k-1}(S)$ under constraints $g(S) = 0$ and $f_k(S) = 0$
- O'_k : to minimize $(f_{k-1}(S), \sum_{j \leq k-2} f_j(S))$ under constraints $g(S) = 0$ and $f_k(S) = 0$

We observe that R_k is a feasibility problem; therefore solving R_k ends with a success or a failure, depending on wether a solution is found or not. During the feasibility phase, subproblem R_k will be solved for $k = 11$ and then, in case of success, for $k = 10$ and so on for the successive decreasing values of k. This iterative process stops when the search fails to solve a particular R_k or if a solution is found for R_0. There are now two possibilities at the end of the feasibility phase. If no solution is found to subproblem R_{11}, it means that no feasible solution is found to the FAPP and the search stops. In the opposite case, we denote by k_{min} the smallest value of k for which a solution has been found and the optimization phase is then performed consisting in solving $O_{k_{min}}$ and then $O'_{k_{min}}$.

Problem R_k can be modelled as a constraint satisfaction problem (CSP), i.e. a problem defined by a set of variables, a set of associated domains and a set of constraints [10]. Such problems are known to be very hard to solve exactly and one generally has to resort to heuristic approaches, in particular when dealing with large instances as is the case here. Among those techniques, local search methods such as tabu search have proven to be very effective [6]. They consist in constructing a configuration (assignment of values to variables) and then modifying it iteratively in order to reduce a cost function that represents the number of violated constraints. O_k and O'_k are constraint satisfaction optimization problems (CSOP), that is CSPs to which an objective function has been added. These can be solved by analogous techniques except that the cost function has now to take the objective into account.

To solve the subproblems R_k, O_k and O'_k, we use local search and tabu. Therefore, each subproblem will now be formulated as an optimization problem (with a set of configurations and an evaluation function) and a neighborhood function between the configurations. Moreover, we define some heuristics in order to restrict the neighborhood (candidate list strategy). Note that these different features are identical for the three problems except for the definition of the cost function.

2.3 Search space and evaluation function

In the following, a *variable* X_i will represent the couple $(p_i,\ f_i)$ to be assigned to link i, and $Dom(X_i)$ the set of possible couples – considering the frequency domain and the polarity domain of link i. A configuration in the search space will be any potential solution or, equivalently, any possible assignment of the variables. In the following, S_{curr} will denote the current configuration in our local search algorithm and $S_{curr}(X_i)$ the value of variable X in S_{curr}.

For R_k, the evaluation function to be minimized is simply defined as the number of violated constraints $g + f_k$. For O_k, the function to be minimized is the weighted sum of two terms penalizing constraint violations: $g + f_k$, that penalizes IC and CEM violations at level k, and f_{k-1}, that penalizes lower degree CEM violations. A weight $\alpha >> 1$ is applied to the first term in order to encourage the procedure to satisfy first the IC and the current level CEM constraints. An analogous principle is used to define the cost function in problem O'_k. The evaluation functions of the subproblems are the following:

- $R_k : g + f_k$
- $O_k : \alpha * (g + f_k) + f_{k-1}$

- O'_k : $\alpha^2 * (g + f_k) + \alpha * f_{k-1} + \sum_{j:j \leq k} f_j$

2.4 Neighborhood function

To solve the CSP by local search, one generally uses an elementary neighborhood structure where two configurations are neighbors if they differ by the value of a single variable. Equivalently, one defines a move as changing the value of a single variable [6]. To solve the FAPP, we define a specialized neighborhood, where a move consists in changing the value of one or two variables. We also tested the elementary neighborhood but our specialized neighborhood turned out to be significantly more efficient.

In the following, the couple $< X, v >$ indicates that the value of variable X is replaced in the current configuration by a new value $v \in Dom(X) - \{S_{curr}(X)\}$. Considering the definition of a variable, note that v represents a couple polarity-frequency and, therefore, the transformation $< X, v >$ may correspond to changing the frequency of X or its polarity or both of them.

The underlying idea of the specialized neighborhood we propose is linked to a particular set of imperative constraints, namely the imperative constraints, frequency equality or CI-FE. Recall that such a constraint imposes a pre-defined distance to be respected between two frequencies: $|f_i - f_j| = D$. Let us suppose now that one of the two variables, say $X_i = (p_i, f_i)$, is fixed; therefore f_i is fixed and there are at most two possible consistent values for f_j: $f_i - D$ and $f_i + D$. Consequently, there are at most 4 consistent values for $X_j = (p_j, f_j)$. In that sense, CI-FE constraints are very strong – much stronger than the constraints of any other type in the FAPP, including EMCs. The second remark is linked to the structure of the (undirected) graph of CI-FE constraints – where each variable X_i is represented by a node i and a CI-FE constraint between f_i and f_j by an edge (i, j). Now, if one analyzes the 40 proposed instances (see [4]), one can note that, except for a few variables that are either isolated (*i.e.* no CI-FE applies to them) or linked to more than one other variable (i.e. several CI-FE apply simultaneously), all variables are generally linked to exactly one other variable in the graph. By using a greedy algorithm to build a (sub-optimal) maximal coupling in the graph, we can partition the set of variables into clusters of two elements (pairs of variables) or single element (singletons). A move now consists in re-assigning at the same time 1 or 2 variables in a cluster. In addition, if the cluster is a doubleton, we impose the CI-FE that links the two variables to be satisfied after the execution of the move:

1. *Singleton* $\{X\}$: a move is any transformation $< X, v >$
2. *Cluster* $\{X, Y\}$, where the CI-FE constraint between X and Y is denoted by C_{XY}: possible moves are
   - any transformation $< X, v >$ such that $(X = v, Y = S_{curr}(Y))$ satisfies C_{XY}
   - any transformation $< Y, w >$ such that $(X = S_{curr}(X), Y = w)$ satisfies C_{XY}
   - any pair of transformations $< X, v >$ and $< Y, w >$ such that $(X = v, Y = w)$ satisfies C_{XY}

Hence the neighborhood of a given configuration comprises all configurations that can be reached through a move.

2.5 Critical variables and values

Given the size of the neighborhoods to be explored, it is clear that considering all possible moves at each iteration would be very time consuming. In order to make the search process more efficient, our local search algorithm will only consider subsets of moves that are considered *a priori* as being among the more promising (i.e. a *candidate list*). A move will be considered promising according to the following definitions of *critical variables* and *critical values*. We define a variable as critical if it is involved in a constraint violated in the current configuration.

We also introduce the notion of critical value in order to take into account the specificity of the problem, in particular the presence of EMCs. Considering a particular variable X, a value $v = (p, f) \in Dom(X) - \{S_{curr}(X)\}$ is said critical if one of the two following conditions holds:

1. (p, f) is an extreme value in the domain,
2. there exists a constraint C and a value $(p, f') \in Dom(X)$ adjacent to (p, f) such that (p, f) satisfies C and (p, f') does not satisfy C.

A value $(p, f) \in Dom(X)$ is an extreme value of X if $(p, g) \notin Dom(X), \forall g > f$ or $(p, g) \notin Dom(X), \forall g < f$. Two values $(p, f), (p', f') \in Dom(X)$ are said adjacent if $p = p'$ and there does not exist any $(p, g) \in Dom(X)$ such that $f < g < f'$ or $f' < g < f$. For example, if $Dom(X) = \{(-1, 5), (-1, 10), (-1, 15), (-1, 20), (-1, 25), (-1, 30), (-1, 35), (1, 10), (1, 15), (1, 20)\}$, then the extreme values of $Dom(X)$ are $(-1, 5), (-1, 35), (1, 10)$ and $(1, 20)$. Given S_{curr}, if constraint C forbids values $(-1, 15)$ and $(-1, 20)$ from being assigned to X, then $(-1, 5)$ and $(-1, 20)$ are also critical values, due to condition 2.

To build the candidate list of moves, we first consider the set of critical variables and select randomly a maximum fixed number of candidate variables among them. Candidate moves are now the moves where at least one candidate variable is modified and where the new value of this variable is a critical value.

3 Solution procedure

In the former section, we proposed a strategy for solving the FAPP based on the definition of different subproblems (R_k, O_k and O'_k) and a local search approach to tackle each of these subproblems that included the definition of an evaluation function, a neighborhood and some heuristics to determine a candidate list of moves. We now present the specific procedure used to solve these subproblems which is composed of a tabu search (TS) meta-heuristic, some diversification techniques and a pre-processing that is used to reduce the size of the search space.

3.1 Tabu mechanism and the aspiration criterion

The *tabu mechanism* we use records couples $< variable, value >$ (see [6]). As long as a particular couple $< X, v >$ is tabu, it is forbidden to assign value v to variable X. At each iteration of the tabu search procedure, a move is performed and one or two variables are modified. Then, for each variable X modified by the move, couple $< X, S_{curr}(X) >$ is classified tabu for a fixed

number of iterations. At each iteration, the candidate list is built and the best non tabu move is chosen in the list (breaking ties randomly). However, a tabu move can eventually be chosen if it leads to a configuration whose cost is strictly inferior to the one of the best solution found during the search (this way of removing the tabu status of a move considering its exceptional quality is called the simple *aspiration criterion* in tabu search jargon).

In order to minimize the risk of cycling and to increase the robustness of the TS procedure, the number of iterations during which a couple remains tabu (or tabu tenure) is varied dynamically during the search process. The principle is to periodically select a visited configuration and to measure the distance between two selected configurations (the distance between two configurations is defined as the number of variables which are assigned different values in these two configurations). Then the tabu tenure is slightly decreased or increased depending on whether the measured distance is larger or smaller than a given threshold. In our experiments, the period equals the number of variables and the threshold is a percentage of the number of variables (according to our experiments, suitable values for this parameter are comprised between 10% and 25%).

3.2 Diversification techniques

In our preliminary experiments, we observed that on some hard instances the basic tabu procedure was not robust enough. The general behavior observed in those cases was that the cost function tended to decrease initially and then to get stuck at a particular level. To overcome this difficulty, we introduced a general diversification technique that we call Adaptive Jumping Procedure (AJP).

The principle of AJP is very general; it is based on the use of two ingredients that are (1) a LS operator and (2) a so-called jumping operator whose role is to *jump* from an initial position in the search space to a new one. In AJP, the LS operator can be any kind of LS technique, for example tabu search but also, possibly, a simple descent. The jumping operator can be seen as a super-mutation that consists in choosing randomly a fixed proportion of the variables and changing their values randomly - but other completely different operators could alternatively be used. An important point is that the amplitude of the diversification is set by a parameter. The principle of AJP is that the *size* of the jump will be adjusted automatically according to the history of the search process. The AJP procedure is defined below:

Procedure AJP(S_0,p_{init})

1. $S_\star = LocalSearch(S_0)$
2. $p = p_{init}$
3. repeat
   (a) $S = Jump(S_\star, p)$
   (b) $S = LocalSearch(S)$
   (c) if $(f(S) == f(S_\star))$, increase p
   (d) if $(f(S) > f(S_\star))$, decrease p

(e) if $(f(S) \leq f(S_\star)), S_\star = S$

S_0 is the initial configuration used for the search and p_{init} is the initial value of the jump amplitude parameter p. $S_\star$ represents the best solution found so far by the overall search and is referred to as the *reference configuration*. AJP is an iterative procedure and one iteration consists in performing a jump from the current configuration and then a LS chain. The quality or performance of the configuration obtained after a full iteration is then compared to that of the reference configuration $S_\star$ and the value of the jump amplitude is adjusted accordingly. There are three possible cases. If the new configuration has the same performance as $S_\star$, then this may indicate that the search stayed in the same region of the solution space and therefore the amplitude p of the jump should be increased to encourage a wider search. If the performance is degraded, then this could mean that the process is jumping too far, i.e. that we are modifying too much of the reference configuration and therefore destroying its structure. Hence, p should be decreased. Finally, if the quality of the resulting solution is improved, the value of p is unchanged. If the performance is not degraded ($f(S) \leq f(S_\star)$), then the new configuration becomes the new reference configuration. Note that we do not allow any degradation of the reference solution. The AJP procedure stops if a solution good enough has been obtained (typically if the optimum is known and reached) or if a limit is reached (time or number of iterations).

3.3 Filtering pre-processing

Eliminating values from the domains of the variables that we are sure can not appear in a feasible solution will reduce the size of the search space and, therefore, should make it easier to solve the problem. To do this we implemented a pre-processing based on a Constraint Programming technique called arc-consistency (see [2, 11]) which is used in a constrained problem (CSP or CSOP) in order to eliminate some values from the domains of the variables. A value will be eliminated if it is proven that it can not belong to any consistent solution (i.e. a solution that satisfies all constraints). Consider two variables X and X' linked by a constraint C; a value v of X is said inconsistent for C if there exists no v' in the current domain of X' such that v can be assigned to X and v' to X' while respecting constraint C. Therefore, if v is inconsistent for C, it is clear that v can not belong to a feasible solution of the problem. The principle of arc-consistency algorithms is to detect such locally inconsistent values, to eliminate them from the domain of the variable and to "propagate" this kind of domain reduction to the other variables as explained hereafter.

The algorithm we use consists in memorizing a set of so-called active constraints. At the beginning of the filtering algorithm, all constraints are active. The algorithm is iterative and each iteration consists in choosing an active constraint, in treating it and in removing it from the set of active constraints. To treat C consists in removing all inconsistent values from the domain of the two variables. Then, if the domain of one of the two variables has been reduced, all constraints that deal with this variable become active (except C). The procedure stops when either the domain of one variable becomes empty, in which case we deduce that the problem is inconsistent and there does exist any feasible solution, or the set of active constraints is empty, in which case we cannot reduce the variable domains any further.

Note now that R_k, O_k and O'_k all contain the same set of constraints that are the imperative constraints, but also all EMCs at level k. The filtering algorithm $Filtering(k)$ consists in applying arc-consistency by considering this last set of constraints. In some cases, the procedure is able to prove the problem to be inconsistent at level k and, therefore, there is no consistent solution to problems R_k, O_k and O'_k. Otherwise, before applying the tabu procedure to solve R_k, O_k and O'_k, we set the domains of the variables according to the values found by $Filtering(k)$.

3.4 Algorithm description

The scheme of our tabu algorithm is presented below:

1. Set k to the feasibility level of the best solution found so far; **if** $k = 0$, **go to** step 4 *(Optimization)*.
2. Apply arc-consistency filtering algorithm to level $k-1$, *Filtering(k-1)*; **if** the problem is proven unfeasible, **go to** step 4 *(Optimization)*.
3. Reduce the domains according to $k-1$; Apply LS to solve problem R_{k-1}; **if** a solution is found, **go to** step 1.
4. Reduce the domains according to level k.
5. Apply LS to solve problem O_k; **if** a *(k-1)*-feasible solution is found, **go to** step 1.
6. Apply LS to solve problem O'_k; **if** a *(k-1)*-feasible solution is found, **go to** step 1.

In the following, k constantly represents the feasibility level of the best solution found so far. According to the principles presented in section 3.2, the algorithm begins with a feasibility phase. In this phase, a *(k-1)*-feasible solution is searched for by applying first the arc-consistency pre-processing (step 2) and then local search (step 3). If successful, then k is updated and steps 1 through 3 are reiterated with the new (and smaller) value of k. If not, the optimization phase is performed for the current value of k by reducing the domains according to level k (step 4) and then solving sub-problem O_k and then O'_k. Note that, by solving O_k or O'_k, a *(k-1)*-feasible solution can also be found. In such a case, the feasibility phase is reiterated by going back to step 1.

As required by the ROADEF Challenge, the stopping criterion for the overall procedure corresponds to an allotted maximum computing time. We use two additional parameters to limit the percentage of time spent in the local search processes of steps 3 and 5, the remainder being devoted to step 6.

The arc-consistency procedure (step 2) has been described in section 3.3. The local search procedure used to solve problem R_{k-1} (step 3) uses the strategy of the Adaptive Jumping Procedure (AJP) presented in section 3.2. The first configuration is built randomly. The LS operator used by AJP is the tabu algorithm obtained by combining the principles and procedures described in sections 2.3 to 3.1. The tabu operator stops if a *(k-1)*-feasible solution is found or if a predetermined maximum number of iterations without improving the best solution is reached. The jumping procedure is a random super-mutation; it consists in randomly

selecting a fixed percentage p of variables and then assigning a random value from their domain to each of them - parameter p is the jump amplitude parameter of the AJP. The initial value of parameter p is 10% of the variables; to increase or decrease p, we add or subtract 10%. The AJP procedure stops when a maximum time limit is reached (see below).

The local search procedures to solve problems O_k and O'_k (steps 5 and 6 respectively) also use the AJP strategy. Each procedure is initialized by using the best k-feasible solution found so far.

4 Computational results

The instances used in our experiments are those of the ROADEF Challenge (see [4]). Three series of instances were proposed during the successive phases of the Challenge, counting 15, 15 and 10 instances respectively, for a total of 40. Their sizes is varied, ranging from around 200 to 3,000 variables, 118 to 15,664 imperative constraints and up to 2,087,947 frequency domain values. The instances are identified by a label fxx_yyyy where xx represents the index of the instances (01 to 40) and $yyyy$ the number of links – for example, the first instance called $f01\_0200$ has 200 links.

We present below two series of results obtained on these instances: the first one was obtained by the algorithm used during the Challenge while the second was obtained by the algorithm presented above. Both algorithms are identical except that the version of the Challenge did not include the filtering technique; therefore, the arc-consistency procedure performed at step 2 is omitted, as well as the resulting domain reductions mentioned in steps 3 and 4.

The same parameters were used in all the experiments reported below. The allotted cpu time is one hour. This total time is split as follows: 70% for the feasibility problems R_k, 20% for the first optimization subproblems O_k, and the remaining 10 % being spent on solving O'_k. In the tabu operator, the maximum number of iterations without improving the best solution is 10,000. The parameter used in the dynamic tabu tenure adjustment mechanism is set to 15% (see section 3.1). In AJP, the initial value of parameter p is 10%; the value used to increase or decrease parameter p is also 10%.

All programs were coded using the C++ programming language. All times reported are in cpu seconds. With respect to time, note that the ROADEF Challenge results were obtained by the Challenge organizers with a single one hour cpu time run on a personal computer equipped with a Pentium III 600 MHz processor. The other results reported hereafter were obtained on SUN workstations with UltraSparcII 400 MHz processors.

4.1 Challenge results and further experiments without filtering

The results presented in this section are those obtained by the algorithm of the Challenge, that is without the arc-consistency filtering procedure. The first columns of table 1 report the instance label, the best results found during the Challenge (considering all participants) and the results of our algorithm during the Challenge. For each instance we report the best feasibility level k, the number of violated constraints at level $k-1$ and the total number of violated constraints (for levels $k-2$ and down) under heading *Sum*. The last portion of the table

reports the results of some additional experiments we carried out to evaluate more precisely the robustness of our approach. We performed 10 independent runs of the algorithm (with different seeds) and observed the best solution found over these runs as well as the distribution of the best feasibility level obtained. This distribution is described here by the *Best* value found followed between parenthesis by the number of times it was achieved over the 10 runs *(Nb)* and the *Worst* k obtained.

If we look at the results obtained by our algorithm with respect to feasibility, we can observe the following. Over the 40 instances, the algorithm systematically finds the best known k value 30 times, it finds it at least once out of the 10 runs for 5 instances ($f12$, $f13$, $f14$, $f29$, and $f35$) and never finds the best k in 5 cases ($f27$, $f30$, $f37$, $f38$, and $f40$). If we now consider the column reporting the *Worst* result found over the 10 runs, we can see that among the 5 instances for which the best k is sometimes reached, there is one case, $f35$, where the method is stuck at $k = 11$ at least once (i.e. unable to find a solution for $k = 10$ while one is known to exist). For those where we never find the best k, there are three cases in which the algorithm is always stuck ($f27$, $f30$, and $f40$). In the two other instances ($f37$ and $f38$), the method is sometimes stuck. Let us now consider the full objective, that is feasibility level k and optimization of violated constraints at levels $k - 1$ and lower, and compare our best results with the best of the Challenge. The method improves on the best result of the Challenge for 16 instances and equals it for 15 others.

We now recall the results obtained by the 4 most efficient algorithms participating in the Challenge (B: our algorithm, C: Caseau, G: Gavranovitch, V: Vasquez). Table 2 summarizes these results. The 3 groups of columns present for each method the number of instances for which: (1) it obtained the best solution (feasibility level and optimization of unsatisfied constraints) among all Challenge participants; (2) it reached the best known value of k; (3) it was stuck at level $k = 11$. We observe that our algorithm was able to find the best known solution for 25 instances out of the 40, *versus* 1 to 5 for the other leading algorithms. Considering this criterion, our algorithm obtained by far the best results in the Challenge. Now with respect to the feasibility level alone, our method did not perform as well, finding the best k value 32 times *versus* 35 and 37 times for algorithms C and V respectively. In particular, our algorithm stayed trapped at $k = 11$ on 5 instances compared to 0 and 1 cases for C and V.

In summary we observe that our algorithm obtained somewhat contrasted results depending on the instances. On the one hand, it obtained very good results on a large majority of them. For these instances it proved to be very efficient compared to the other participants as well as robust in terms of its ability to reach the best known feasibility level. On the other hand, the procedure did not perform so well and even did quite poorly on a few of them when compared to algorithms C and V. We introduced the filtering technique after the Challenge in order to improve our results on this limited number of instances.

4.2 Algorithm with filtering techniques

We now discuss the impact of introducing the filtering technique described in section 3.3. We are first going to analyze the effect of the filtering procedure $Filtering(k)$ in terms of domain reduction and its ability to prove infeasibility. Then we will compare the results of the algorithm *with* and *without* filtering.

Instance	ROADEF Challenge						Further experiments - 10 runs				
	Best result			Our result			Best result			k	
	k	$k-1$	Sum	k	$k-1$	Sum	k	$k-1$	Sum	Best(Nb)	Worst
f01_0200	4	4	56	4	4	56	4	4	37	4 (10)	4
f02_0250	2	7	86	2	7	86	2	7	20	2 (10)	2
f03_0300	7	10	165	7	10	341	7	10	231	7 (10)	7
f04_0300	1	31	0	1	31	0	1	16	0	1 (10)	1
f05_0350	11	1	186	11	1	372	11	1	268	11 (10)	11
f06_0500	5	12	246	5	12	246	5	12	250	5 (10)	5
f07_0600	9	22	682	9	22	714	9	22	732	9 (10)	9
f08_0700	5	16	266	5	16	266	5	16	247	5 (10)	5
f09_0800	3	28	195	3	28	195	3	28	210	3 (10)	3
f10_0900	6	18	475	6	18	475	6	18	442	6 (10)	6
f11_1000	8	8	1015	8	8	1015	8	8	784	8 (10)	8
f12_1500	2	62	1310	3	83	1698	2	3	44	2 (4)	5
f13_2000	3	49	2003	3	49	2003	3	48	1377	3 (8)	4
f14_2500	4	35	3485	4	35	3485	4	39	3750	4 (7)	5
f15_3000	5	15	1569	5	15	1569	5	15	1415	5 (10)	5
f16_0260	11	5	56	11	5	56	11	5	56	11 (10)	11
f17_0300	4	4	34	4	4	34	4	4	34	4 (10)	4
f18_0350	8	4	55	8	4	55	8	4	55	8 (10)	8
f19_0350	6	2	51	6	2	51	6	2	51	6 (10)	6
f20_0420	10	5	97	10	5	97	10	5	97	10 (10)	10
f21_0500	4	2	10	4	2	10	4	2	10	4 (10)	4
f22_1750	7	15	187	7	15	187	7	15	187	7 (10)	7
f23_1800	9	16	187	9	16	187	9	16	187	9 (10)	9
f24_2000	7	6	71	7	6	71	7	6	71	7 (10)	7
f25_2230	3	7	32	3	7	32	3	7	32	3 (10)	3
f26_2300	7	9	74	7	9	74	7	9	74	7 (10)	7
f27_2550	5	4	20	11	4	64	11	4	64	11 (10)	11
f28_2800	3	13	32	3	13	32	3	13	32	3 (10)	3
f29_2900	6	25	212	6	25	239	6	25	216	6 (9)	7
f30_3000	7	13	148	11	1166	12029	11	22	15421	11 (10)	11
f31_0400	5	4	1180	5	4	1180	5	4	1143	5 (10)	5
f32_0550	6	5	71	10	52	1739	6	4	39	6 (10)	6
f33_0650	5	7	66	5	7	66	5	7	60	5 (10)	5
f34_0750	4	2	46	4	2	46	4	2	46	4 (10)	4
f35_1500	6	16	431	7	3	1280	6	7	148	6 (6)	11
f36_2000	7	19	1451	7	99	2153	7	8	290	7 (10)	7
f37_2250	5	51	1288	11	3	12229	6	1	228	6 (1)	11
f38_2500	3	14	174	11	79	14058	10	7	8287	10 (1)	11
f39_2750	3	356	2844	3	356	2844	3	344	5360	3 (10)	3
f40_3000	4	64	1252	11	39	16755	11	16	16277	11 (10)	11

Table 1: Results without filtering

	Best results				Best k				Stuck (k=11)			
	B	C	G	V	B	C	G	V	B	C	G	V
Phase 1 (f01 to 15)	11	.	.	1	14	11	6	12	.	.	4	.
Phase 2 (f15 to 30)	10	1	2	.	13	15	15	15	2	.	.	.
Phase 3 (f31 to 40)	4	.	3	2	5	9	7	8	3	.	.	1
Total	25	1	5	3	32	35	28	37	5	0	4	1

Table 2: ROADEF Challenge - finalists aggregate results

Instance	k		Initial	Remaining domain for each feasibility level k (%)											Time
	Min	Inf	dom size	10	9	8	7	6	5	4	3	2	1	0	
f01_0200	4	1	26963	78.7	72.7	67.2	62	60.3	55.3	53.2	49.1	44.5	—	—	0
f02_0250	2	1	36618	87.3	74.5	66.5	60	53.3	51.4	48.3	43.6	40.9	—	—	1
f03_0300	7	6	53536	76.5	65.5	59	52.9	—	—	—	—	—	—	—	1
f04_0300	1	0	61762	86.3	80.7	77.4	72.8	69.8	66.1	59.6	56.7	48.5	37.1	—	1
f05_0350	11	7	79311	88.7	80.6	68.7	—	—	—	—	—	—	—	—	1
f06_0500	5	4	108024	87.4	75.1	69.9	63.2	55.3	49.4	—	—	—	—	—	2
f07_0600	9	8	109658	79.8	64.1	—	—	—	—	—	—	—	—	—	1
f08_0700	5	4	134020	88.8	82.3	76.5	72.3	66.8	62.6	—	—	—	—	—	2
f09_0800	3	2	121824	79	70.4	66	62	59.2	56.3	52	43.9	—	—	—	3
f10_0900	6	5	197665	84	77.5	71.4	67.6	62	—	—	—	—	—	—	3
f11_1000	8	7	294634	78.2	65.2	53.2	—	—	—	—	—	—	—	—	4
f12_1500	2	1	436967	85.4	67.1	58.4	53.8	50.6	47.4	45	42.8	39.1	—	—	11
f13_2000	3	2	320494	87.8	80.3	69.9	63.2	59.1	55.9	51.9	46.8	—	—	—	9
f14_2500	4	3	774322	81.1	68.2	58.9	51.6	47.6	45	41.7	—	—	—	—	18
f15_3000	5	4	515606	93.9	88.2	80.9	74.2	67.6	60.6	—	—	—	—	—	11
f16_0260	11	10	47293	—	—	—	—	—	—	—	—	—	—	—	0
f17_0300	4	3	64034	1.5	1.4	1.4	1.4	1.4	1.4	1.4	—	—	—	—	0
f18_0350	8	7	73016	3	2.8	2.8	—	—	—	—	—	—	—	—	0
f19_0350	6	5	201074	2.5	2.3	2.1	2.1	2	—	—	—	—	—	—	1
f20_0420	10	9	87077	2.3	—	—	—	—	—	—	—	—	—	—	0
f21_0500	4	3	113594	8.2	7.4	7.2	7.1	7	6.8	6.8	—	—	—	—	1
f22_1750	7	6	813037	1.9	1.7	1.6	1.6	—	—	—	—	—	—	—	7
f23_1800	9	8	455735	0.8	0.8	—	—	—	—	—	—	—	—	—	5
f24_2000	7	6	567396	1.7	1.5	1.5	1.5	—	—	—	—	—	—	—	5
f25_2230	3	2	610084	5.7	5.4	5.3	5.2	5.1	5.1	5	5	—	—	—	6
f26_2300	7	6	635123	2.2	2	2	1.9	—	—	—	—	—	—	—	6
f27_2550	5	4	588188	18.2	17.3	16.8	16.4	16.2	16.1	—	—	—	—	—	7
f28_2800	3	2	2087947	3.9	3.7	3.6	3.6	3.4	3.4	3.4	3.4	—	—	—	17
f29_2900	6	5	1477634	1.2	1.1	1.1	1.1	1.1	—	—	—	—	—	—	13
f30_3000	7	6	1942250	4.8	4.6	4.5	4.4	—	—	—	—	—	—	—	21
f31_0400	5	2	273538	68.7	67.7	66.5	65.6	64.6	63.1	60.6	58.5	—	—	—	5
f32_0550	6	5	448436	3.5	3	2.8	2.6	1.6	—	—	—	—	—	—	4
f33_0650	5	4	233788	5.7	5.3	5	4.9	4.7	4.1	—	—	—	—	—	2
f34_0750	4	3	329841	7.7	7.1	6.6	6.3	6.1	6	5.8	—	—	—	—	3
f35_1500	6	5	844907	4.4	4	3.8	3.6	3.5	—	—	—	—	—	—	8
f36_2000	7	6	750979	5.5	5.1	4.8	4.5	—	—	—	—	—	—	—	7
f37_2250	5	4	1531733	3.9	3	2.5	2.2	1.9	1.6	—	—	—	—	—	18
f38_2500	3	2	1460508	3.7	3.1	2.7	2.3	1.9	1.6	1.5	1.4	—	—	—	25
f39_2750	3	1	1343881	76.3	75.3	74.3	73.4	72.3	71.3	70.2	68.9	65.5	—	—	36
f40_3000	4	3	1873230	4.7	3.7	3.2	2.8	2.5	2.3	2.1	—	—	—	—	28

Table 3: Results of the filtering procedure

Table 3 presents the results obtained by the filtering procedure alone. For each instance we first indicate under the headings k *Min* the best feasibility level found during the Challenge (all participants considered) and under k *Inf* the largest value of k for which the filtering procedure was able to prove infeasibility. The rest of the table reports the initial domain size (the sum of the domain sizes of all variables), the portion of the domain remaining (in percentage points) after the filtering and resulting domain reduction for each feasibility level k ($k = 0, ..., 10$), and finally, the total time required to perform the filtering over all feasibility levels (in cpu seconds).

As was expected, one can observe that the "efficiency" of the filtering increases when the constraints become stronger which of course translates in smaller and smaller percentage figures as k decreases. For the smaller values of k, the filtering algorithm is able to prove that the instance is infeasible which we indicate by character "—" in the table. From the table we can also see that the percentage of remaining domain values, for the smallest value of k for which the filtering does not prove infeasibility, varies significantly between the three phases: it ranges from 37.1% to 88% in Phase 1, from 0.8% to 16.1% in Phase 2 and from 1.4% to 68.9% in Phase 3. These are quite impressive reductions in domain size and therefore in the size of the problems to be solved which should *a priori* facilitate the task of the tabu search procedure.

Finally, when comparing the values of $k\ Min$ and $k\ Inf$, we observe that these values differ by only one unit for all but 4 instances ($f1$, $f5$, $f31$, and $f39$). For these 36 instances, this proves both that the best feasibility levels obtained during the Challenge are indeed optimal and that the filtering procedure was able to prove infeasibility for each infeasible instance.

Table 4 presents the results obtained by our tabu search algorithm when the filtering procedure is included and used as a pre-processing at each feasibility level. We only consider here the 10 instances for which the original version was not able to systematically reach the best known feasibility level. Considering the 3 instances of Phase 1, the results have not been improved by the filtering. In fact, there is a degradation of the results for all 3 instances. With regards to the 7 instances of Phases 2 and 3, we observe a considerable improvement in the results: for 4 of them, the filtering makes it possible to systematically reach the optimum value of k over the 10 runs, for 2 others we obtain the best k 9 times and for the last one 3 times.

We note that the improvement of the results occurs for instances coming from Phases 2 and 3 in which the domain reduction resulting from the filtering is very important (84 to 99 %). On the other hand, we do not observe improvements for the Phase 1 instances for which the domain reductions were much smaller (50 to 60 %). These results seem to indicate that the filtering should not automatically be used with the proposed tabu search procedure, but should instead be considered as an additional tool to tackle problematic instances.

5 Conclusion

In this paper, we have proposed an efficient tabu search algorithm to solve the FAPP. This algorithm incorporates some original features: the decomposition of the problem (search strategy), the neighborhood function, the candidate list, and the technique to adjust the tabu tenure. We also implemented a pre-processing procedure based on arc-consistency.

Some of these techniques are problem-specific as the neighborhood function and the definition of critical values used in the candidate list, although the latter could also be used for other

Instance	Challenge Best result			Rem dom	Without filtering					With filtering				
					k		Best result			k		Best result		
	k	$k-1$	Sum	(%)	Best(Nb)	Worst	k	$k-1$	Sum	Best(Nb)	Worst	k	$k-1$	Sum
f12_1500	2	62	1310	39.1	2 (4)	5	2	3	44	4 (2)	6	4	8	1385
f13_2000	3	49	2003	46.8	3 (8)	4	3	48	1377	3 (6)	4	3	54	1391
f14_2500	4	35	3485	41.7	4 (7)	5	4	39	3750	4 (3)	6	4	31	2232
f27_2550	5	4	20	16.1	11 (10)	11	11	4	64	5 (10)	5	5	4	21
f29_2900	6	25	212	1.1	6 (9)	7	6	25	216	6 (10)	6	6	25	216
f30_3000	7	13	148	4.4	11 (10)	11	11	22	15421	7 (9)	8	7	13	747
f35_1500	6	16	431	3.5	6 (6)	11	6	7	148	6 (10)	6	6	7	132
f37_2250	5	51	1288	1.6	6 (1)	11	6	1	228	5 (9)	11	5	6	107
f38_2500	3	14	174	1.4	10 (1)	11	10	7	8287	3 (3)	10	3	42	548
f40_3000	4	64	1252	2.1	11 (10)	11	11	16	16277	4 (10)	4	4	8	102

Table 4: Comparative performance: with vs without filtering

frequency assignment problems. Some other techniques are general an potentially applicable to a large number of new problems. This is the case for the mechanism proposed to update the tabu tenure and the diversification strategy called Adaptive Jumping Procedure (AJP).

We tested our tabu algorithm on the 40 instances of the ROADEF Challenge 2001 and obtained very good results. The algorithm was able to reach the best known feasibility level for all instances and also to find or improve on the best known solutions of the Challenge for a majority of the instances (28 out of the 40).

The key features of the algorithm seem to be the neighborhood, the candidate list strategy and the filtering pre-processing. We report some test results that show the efficiency of the filtering to solve some instances. It would now be interesting to analyze the impact of other key features of the algorithm like the neighborhood (that should be compared to the elementary one) and the candidate list strategy. Another research avenue is to test the AJP for new problems.

References

[1] Aarts, E.H.L., Lenstra, J.K. (Eds.), *Local Search in Combinatorial Optimization*, John Wiley & Sons, 1997.

[2] Bessiere, C., Regin, J.C., "Refining the Basic Consistency Propagation Algorithm", 17th International Joint Conference on Artificial Intelligence (IJCAI´01), p. 309-315, Seatle Washington, USA, 2001.

[3] Cabon, B., Givry, S. D., Lobjois, L., Schiex, T., and Warners, J. P. , "Benchmarks Problems: Radio Link Frequency Assignment", *Constraints*, 4:79-89, 1999.

[4] Cung, V.D., Defaix, T., "Quelques problèmes d´allocation de fréquences et le challenge ROADEFF'2001", JNPC´2001, p. 9-19, 2001.

[5] FAP Bibliography site, http://fap.zib.de/biblio/content.html.

[6] Galinier, P., Hao, J.K., "Tabu Search for Maximal Constraint Satisfaction Problems", Proc. of 3rd Intl. Conf. on Principles and Practice of Constraint Programming (CP'97). Lecture Notes in Computer Science 1330 : 196-208, Springer-Verlag, 1997.

[7] Glover, F. and Laguna, M., *Tabu Search*, Kluwer Academic Publishers, 1997.

[8] Hao, J.K., Dorne, R., Galinier, P., "Tabu Search for Frequency Assignment in Mobile Radio Networks", *Journal of Heuristics* 4:47-62, 1998.

[9] Hertz, A., Schindl, D.,Zufferey, N., "Lower Bounding and Tabu Search Procedures for the Frequency Assignment Problem with Polarization Constraints", submitted, 2003.

[10] Mackworth, A.K., "Constraint Satisfaction", in S.C. Shapiro (Ed.) *Encyclopedia on Artificial Intelligence*, John Wiley & Sons, NY, 1987.

[11] Mohr, R., Henderson, T.C., "Arc and Path Consistency Revisited", *Artificial Intelligence*, 28:225-233, 1986

[12] Murphey, R.A., Pardalos, P.M., Resende, M.G.C., "Frequency Assignment Problems", in D.-Z. Du, P.M. Pardalos (Eds.), *Handbook of Combinatorial Optimization Supplement Volume A*, Kluwer Academic Publishers, 1999.

[13] Tiourine, S.R., Hurkens, C.A.J., and Lenstra, J.K., "Local Search Algorithms for the Radio Link Frequency Assignment Problem", *Telecommunication systems*, 13:293-314, 2000.

[14] Tsang, E., *Foundations of Constraint Satisfaction*, Academic Press, 1993.