Tabu Search for Frequency Assignment in Mobile Radio Networks*

Jin-Kao Hao, Raphäel Dorne and Philippe Galinier
LGI2P
EMA-EERIE
Parc Scientifique Georges Besse
F-30000 Nîmes
France
email: {hao, dorne, galinier}@eerie.fr

In the *Journal of Heuristics*, 4: 47-62 (1998), Kluwer Academic Publishers

Abstract

The main goal of the Frequency Assignment Problem in mobile radio networks consists of assigning a limited number of frequencies to each radio cell in a cellular network while minimizing electro-magnetic interference due to the re-use of frequencies. This problem, known to be NP-hard, is of great importance in practice since better solutions will allow a telecommunications operator to manage larger cellular networks. This paper presents a new Tabu Search algorithm for this application. The algorithm is tested on realistic and large problem instances and compared with other methods based on simulated annealing, constraint programming and graph coloring algorithms. Empirical evidence shows that the Tabu algorithm is very competitive by giving the best solutions to the tested instances.

Keywords: Tabu Search, frequency assignment, constraints, optimization

1 Introduction

The Frequency Assignment Problem (FAP) is one of the key applications in mobile radio networks engineering[1]. Although different versions of the FAP can be defined, the main purpose is to assign a limited number of available frequencies to each cell in a mobile radio network while minimizing electro-magnetic interference due to the re-use of frequencies. The difficulty of this application comes from the fact that an acceptable solution of the FAP must satisfy a set of multiple constraints, which impose conflicting objectives. The most severe constraint concerns a very limited radio spectrum consisting of a small number of frequencies (or channels). Telecommunications operators such as France Telecom must cope with a maximum of 60 frequencies for their networks, whatever the traffic volume

*Work supported by the CNET (French National Research Center for Telecommunications) under the grant No.961B134.

[1]The frequency assignment problem studied in this paper applies essentially to widely used cellular networks and does not necessarily apply to other wireless technologies, such as the CDMA.

to be covered, and this, in agreement with national and international regulations. This constraint imposes a high degree of frequency re-use, which in turn increases the probability of frequency interference. Indeed, in addition to this constraint, there are frequency interference constraints which state that frequencies assigned to some cells must satisfy a given separation distance in the frequency domain.

The basic FAP can be shown to be NP-hard in its simplest form because it is reduced to the graph coloring problem [13]. More generally, the problem is equivalent to the so-called "set T-coloring problem" [20]. Therefore, it is very unlikely to find any efficient algorithm for this problem. So far, many heuristic methods have been proposed to tackle the FAP in cellular networks, including graph coloring algorithms (GCA) [11], constraint programming (CP) [1], simulated annealing (SA) [8], artificial neural networks [18, 10], evolutionary algorithms (EAs) [14] and Tabu Search (TS) [15]. Similar techniques have been developed and experimented on a closely related problem called "radio link frequency assignment" in the setting of military application [6, 17, 3, 4].

In this paper, we present a new and highly effective TS algorithm tailored for the frequency assignment problem. This algorithm distinguishes itself from previous algorithms by some important features such as incremental evaluation of solutions, dynamic tabu tenure, candidate list strategy, and co-cell constraints handling. Experiments carried out on realistic and large problem instances (up to 300 cells, 30,000 constraints with 30 frequencies) show that this TS algorithm is very competitive and gives us the best results for the tested instances compared with other methods based on graph coloring algorithms, constraint programming, and simulated annealing.

The paper is organized as follows. In Section 2, the FAP is modeled as an optimization problem. In Section 3, TS is briefly reviewed. In Section 4, the components of the TS algorithm are described. In Section 5, after a review of the test data, experimental results are presented and compared. Conclusions are given in the last section.

2 The Frequency Assignment Problem

2.1 Constraints in the FAP

A cellular network is defined by a set $\{C_1, C_2 ... C_N\}$ of N cells, each cell C_i requiring T_i frequencies. The number T_i, called the traffic of C_i, is determined by an estimation of the maximum number of communications which can simultaneously arise within a cell.

The basic FAP consists of assigning to each cell C_i of the network T_i frequencies taken from a set of available frequencies while respecting a set of frequency *interference constraints*. Since the number of available frequencies is very limited and usually much smaller than the sum of the total traffics of the network, frequencies must be re-used by different cells in an assignment. It is this frequency re-using that may lead to frequency interference.

Interference occurs when two frequencies assigned to a same cell or two adjacent cells[2] are not sufficiently separated. Therefore, the *frequency interference constraints* over a network are divided into *co-cell constraints* and *adjacent-cell constraints*.

- *co-cell constraint*: any pair of frequencies assigned to a radio cell must have a certain distance between them in the frequency domain.

[2]Two cells are adjacent if they emit within a common area even if they are not geographically adjacent.

- *adjacent-cell constraint*: any pair of frequencies assigned to two adjacent cells must be sufficiently separated in the frequency domain.

These interference constraints are conveniently represented by a symmetric *compatibility matrix* $M[N, N]$ where N is the number of cells in the network and each element of M is a non negative integer. Let $f_{i,k}$ denote the value of the kth frequency ($k \in \{1..T_i\}$) of C_i and let $\{1..NF\}$ denote the set of NF available frequency values, then the interference constraints are formulated as follows:

- $M[i, i]$ (i $\in \{1..N\}$) is the minimum frequency separation necessary to satisfy the co-cell constraints for the cell C_i. $\forall$m, n $\in \{1..T_i\}$, m $\neq$ n, $|f_{i,m} - f_{i,n}| \geq M[i, i]$
- $M[i, j]$ (i, j $\in \{1..N\}$, i $\neq$ j) represents the minimum frequency separation required to satisfy the adjacent-cell constraints between two cells C_i and C_j. $M[i, j] = 0$ means there is no constraint between the cells C_i and C_j.
  $\forall$m $\in \{1..T_i\}$, $\forall$n $\in \{1..T_j\}$, $|f_{i,m} - f_{j,n}| \geq M[i, j]$

2.2 Optimization in FAP

Given a network defined by N cells with T_i traffic for each cell C_i and the compatibility matrix M, two basic optimization (minimization) problems can be defined.

The first problem consists of minimizing frequency interference with a fixed number of frequencies. In this case, the optimization problem is defined by a couple *(S,f)* where the search space S is defined by the set of all possible frequency assignments and the cost function f to be minimized is defined as follows. For any s in S, $f(s)$ represents frequency interference, measured by the number of unsatisfied co-cell and adjacent-cell constraints. More formally, one has:

$$f(s) = \sum_{i=1}^{N} \sum_{m=1}^{T_i} \sum_{n=m+1}^{T_i} CO(i, m, n) + \sum_{i=1}^{N} \sum_{j=i+1}^{N} \sum_{m=1}^{T_i} \sum_{n=1}^{T_j} AD(i, j, m, n) \quad (1)$$

$$CO(i, m, n) = \begin{cases} 1 \text{ if } |f_{i,m} - f_{i,n}| < M[i, i] \\ 0 \text{ otherwise} \end{cases} \quad AD(i, j, m, n) = \begin{cases} 1 \text{ if } |f_{i,m} - f_{j,n}| < M[i, j] \\ 0 \text{ otherwise} \end{cases}$$

The second problem consists of minimizing the number of frequencies required in an interference-free assignment[3]. This problem can be dealt with as a series of the first problems as follows.

To minimize the number of frequencies NF used in an assignment, one fixes NF to a sufficiently high value[4] and seeks an interference-free assignment s ($f(s) = 0$) within the above mentioned search space S and the cost function f. If such an interference-free assignment is found, one proceeds with *NF-1* frequencies and so on. This process continues until one can no longer find an interference-free assignment. This minimization process is similar to that used by Hertz and de Werra in their TS algorithm for graph coloring [16].

Let us notice that minimizing the number of frequencies used is of great importance in practice since free frequencies can easily be used for extensions to an existing network.

[3]More generally, the interference-free requirement may be relaxed to allow any interference threshold.
[4]This initial value can be determined with a greedy method.

3 A Brief Review of Tabu Search

This section gives a brief review of Tabu Search, emphasizing the most important features which have been implemented in our TS algorithm. For a complete presentation of TS, the reader is invited to consult the recent book by Glover and Laguna [12].

Tabu Search is a meta-heuristic designed for tackling hard combinatorial optimization problems. Contrary to randomizing approaches such as SA where randomness is extensively used, TS is based on the belief that intelligent searching should embrace more systematic forms of guidance such as memorizing and learning.

TS can be described as a form of neighborhood search with a set of critical and complementary components. For a given instance of an optimization problem characterized by a search space S and a cost function f, a function $N\colon S \rightarrow 2^S$ is first introduced to determine a neighborhood. A typical TS algorithm begins then with an initial configuration s in S and then proceeds iteratively to visit a series of locally best configurations following the neighborhood function. At each iteration, a *best* neighbor $s' \in N(s)$ is sought to replace the current configuration even if s' does not improve the current configuration in terms of the cost function. To avoid the problem of possible cycling and to allow the search to go beyond local optima, TS introduces the notion of *Tabu list*, one of the most important components of the method.

A tabu list is a special short term memory that maintains a selective history H, composed of previously encountered solutions or more generally pertinent attributes of such solutions. A simple TS strategy based on this short term memory H consists in preventing solutions of H from being reconsidered for next k iterations (k, called tabu tenure, is problem dependent). Now, at each iteration, TS searches for a best neighbor from this dynamically modified neighborhood $N(H,s)$, instead of $N(s)$ itself. Such a strategy prevents Tabu from being trapped in short term cycling and allows the search process to go beyond local optima.

When attributes of solutions instead of solutions are recorded in tabu list, some non-visited, yet interesting solutions may be prevented from being considered. *Aspiration criteria* may be used to overcome this problem. One widely used aspiration criterion consists of removing a tabu classification from a move when the move leads to a solution better than the best obtained so far.

TS uses an aggressive search strategy to exploit its neighborhood. Therefore, it is crucial to have special data structures and techniques which allow a fast updating of move evaluations, and reduce the effort of finding best moves. Moreover, *candidate list strategies* may be used to limit the neighbors to be considered at each iteration to a subset of the entire neighborhood. A typical strategy consists of identifying subsets of influential moves, such as those considered promising to lead to improved solutions. A good candidate list strategy combined with an efficient technique for move evaluations is essential for high solution speed and good quality.

There are other techniques available in TS such as intensification and diversification. In this paper, we show that a TS algorithm based on the above mentioned elements may be very effective and robust.

4 A TS Algorithm for FAP

The following notations will be used in the presentation: N the number of cells of a network, NF the number of available frequencies, $\{1..NF\}$ the set of NF frequency values, C_i and T_i a cell and its traffic, $f_{i,k}$ ($f_{i,k} \in \{1..NF\}$) the value of the kth frequency of C_i.

4.1 Components

Configuration, search space and cost function

Given a FAP instance involving N cells, traffics T_i (i $\in \{1..N\}$), a set of frequency values $\{1..NF\}$, and a compatibility matrix $M[N, N]$, a configuration s corresponds to a complete frequency assignment:

$s = < f_{1,1}...f_{1,T_1}...f_{i,1}...f_{i,T_i}...f_{N,1}, f_{N,T_N} >$ such that:

$\forall$ i $\in \{1..N\}$, $\forall$m, n $\in \{1..T_i\}$, m $\neq$ n, $|f_{i,m} - f_{i,n}| \geq M[i, i]$.

In other words, a configuration is a frequency assignment satisfying the co-cell constraints of the given instance. The search space S of the instance is therefore composed of all such configurations.

According to equation (1) (Section 2), for each solution $s \in S$, *f(s)* corresponds to the total number of unsatisfied interference constraints. Since co-cell constraints are taken into account by the configuration s, *f(s)* is now simplified to be the number of unsatisfied *adjacent-cell* constraints. Note that for an interference-free assignment s, *f(s) = 0*.

An alternative approach consists of not imposing the satisfaction of co-cell constraints. However, as explained later, directly integrating co-cell constraints into configurations is an important factor in improving the search efficiency for this application.

Neighborhood and candidate list

Given S, the neighborhood function $N : S \rightarrow 2^S$ is defined as follows: s and $s' \in S$ are neighbors if they are different at the value of a single frequency of a cell. More formally, let $s(f_{i,k})$ denote the value of the kth frequency of the cell C_i in s, then $s' \in N(s)$ if and only if the following condition is verified:

$\exists\,!$ (i,m) $\in \{1..N\} \times \{1..T_i\}$ such that $s(f_{i,m}) \neq s'(f_{i,m})$ and
$\forall$ (j,n) $\in \{1..N\} \times \{1..T_j\}$ (j,n) $\neq$ (i,m), $s(f_{j,n}) = s'(f_{j,n})$

Therefore, a neighbor of s can be obtained by changing the value $f_{i,m}$ of the mth frequency of a cell C_i in s in such a way that the new value always satisfies the co-cell constraint. A move is thus characterized by a triplet $< i, m, v >$, i, m and v being respectively a cell, a frequency of the cell and a frequency value.

The members of *N(s)* are not equally interesting. Indeed, it will be less pertinent to change a frequency value which does not violate any interference constraint. This observation leads us to define the following candidate list strategy:

$V^* = \{s' \in N(s) \mid s'$ and s are different at the value of a *conflicting* frequency[5]$\}$

[5] A frequency of a cell is said to be conflicting if its value violates some interference constraints.

Clearly, the size of V^* varies during the search according to the number of conflicting frequencies and in general is much smaller than $|N(s)|$. This candidate list strategy helps the search to concentrate on influential moves and to avoid irrelevant ones. Moreover, it significantly reduces the number of neighbors to be considered at each iteration.

Incremental evaluation and neighborhood examination

For any solution s, its cost *f(s)* represents frequency interference, measured by the number of unsatisfied interference constraints. Since the number of neighbors to consider may be very high at each iteration, we adopt a fast evaluation method inspired by a technique proposed by Fleurent and Ferland [9]. Let us note $W = \sum_{i=1}^{N} T_i$. The main idea consists of maintaining incrementally in a $NF * W$ matrix δ the *move value* or *cost variation* for each possible move of the current solution s. Each time a move is carried out, the elements of the matrix affected by the move are updated accordingly. Initializing δ takes time $O(NF * W^2)$. After a move, the matrix can be updated in time $O(NF * W)$ in the worst case. A best neighbor in V^* can be searched in time $O(|V^*|)$. Given that $|V^*|$ is smaller then $NF * W$, each iteration takes thus time $O(NF * W)$ in the worst case.

Tabu list and tabu tenure

When the value of a frequency of a cell C_i in a solution s is changed to a new value, the pair $< cell, old\_value >$ is classified tabu for k (tabu tenure) iterations. That is, the old value will not be allowed to be re-assigned to C_i during this period. To implement the tabu list, we use an $N * NF$ matrix T. Each time $< cell, old\_value >$ is added to the tabu list, the corresponding element of T is set to the current iteration number plus tabu tenure k. At any moment, it is easy to verify if a given move is tabu or not by simply comparing the current iteration number with that recorded in the tabu matrix T.

The tabu tenure k is dynamically adjusted by a function defined over the size of the candidate list V^*. More precisely, $k = \alpha|V^*|$ where α is a value from (0,1] and k is bounded by two constants *Min* and *Max.* Since $|V^*|$ varies during the search, so does k for any fixed α. α typically takes values from 0.1 to 0.5. To avoid too large values of k due to the presence of many conflicting cells in a configuration, a constant *Max* is used to limit k. Similarly, a lower bound *Min* is used to prevent k from becoming too small at late stages of the search. *Min* and *Max* are typically fixed at $10 * NF$ and *NF.*

Aspiration criteria

A very simple aspiration criterion is used: the tabu status of a move is cancelled if the move leads to a solution better than the best solution s^* encountered so far.

4.2 Tabu Algorithm

Now that the components of the TS algorithm have been presented, we give below the general algorithm.

Tabu algorithm

```
begin
    $T = 0$ /* initialize tabu matrix */
    $Iter = 0$ /* initialize the iteration counter */
    $s$ = generate() /* generate an initial solution */
    $s^* = s$ /* record the best configuration found so far */
    $f^* = f(s^*)$
    while $Iter < MAX$ do
        $Iter = Iter + 1$
        choose a best neighbor $s' = s + < i, m, new\_v > \in V^*$
        such that $NB > T[i, new\_v]$ or $f(s') < f^*$
        $s = s'$
        $T[i, old\_v] = Iter + k$; /* $< C_i, old\_v >$ becomes tabu */
        update $\delta$ matrix
        update tabu tenure $k$ ($k = \alpha|V^*|$)
        if $f(s) < f^*$ then
            $s^* = s$
            $f^* = f(s^*)$
end
```

Figure 1. Tabu Search Algorithm for the Frequency Assignment Problem

This algorithm can directly be used to deal with the first optimization version of the frequency assignment problem defined in Section 2, i.e. to minimize frequency interference with a fixed number of *NF* frequencies. The algorithm stops and returns the best assignment found after MAX iterations.

This algorithm can also be used to minimize the number of frequencies *NF* required in an interference-free assignment as explained in Section 2.2:

1. apply the above TS algorithm to search an interference-free assignment s ($f(s) = 0$) with *NF* frequencies;
2. decrement *NF*, i.e. $NF \leftarrow NF-1$, and go to step 1 if such an assignment is found;
3. stop and return *NF+1* if no interference-free solution is found within MAX iterations.

Each time the TS algorithm is called with a decremented number of frequencies, an initial configuration using *NF-1* frequencies must be generated. There are two possibilities to do this. First, one can generate a completely new configuration from scratch. Second, one can re-use the last assignment obtained with *NF* frequencies as follows: for each frequency having a value greater than the decremented *NF*, it is given a new value randomly taken from 1 to the decremented *NF*[6].

Note finally that this optimization process can be used to minimize the number of frequencies *NF* required by an assignment with any admissible interference threshold.

4.3 Comments on the TS Algorithm

The current TS algorithm distinguishes itself from a previous TS algorithm [15] by some important features. The first feature concerns the evaluation method. The δ matrix

[6]Given the definition of a configuration, this new frequency value must verify co-cell constraints.

introduced above allows a fast searching of best moves and an efficient updating of move values. Conversely, no data structure is used in [15] to record move values. The evaluation of neighbors is thus expensive and very time-consuming in the previous TS algorithm.

The second point concerns the candidate list strategy. It is well-known that the number and nature of the neighbors visited at each iteration have a great influence on the performance of a TS algorithm. The conflict-based strategy is an informative way to isolate influential moves and is more efficient than the entire neighborhood. Experimental data are presented in Section 5.4. to support this point.

The third point concerns the way co-cell constraints are handled. In the initial problem formulation, both co-cell and adjacent-cell constraints are used to define the cost function (see Section 2). Therefore, both of them can be treated exactly in the same way to evaluate the quality of a configuration. This is the approach taken in [15]. However, a close look at these two types of constraints makes it clear that they are in fact different. In some sense, co-cell constraints are harder to solve than adjacent-cell constraints since co-cell constraints, defined on each pair of frequencies of a same cell, form cliques. As shown in [7], it is more interesting to remove these hard constraints from the cost function and solve them first. This is realized by imposing the satisfaction of co-cell constraints by all configurations (feasibility with respect to these constraints). This technique, called co-cell handling (CCH) in [7], allows to reduce the search space and to simplify the cost function. Experimental data are presented in Section 5.4. to support this point.

Finally, the dynamic tabu tenure is another factor contributing to the efficiency of the algorithm. Indeed, it seems more natural to allow the tabu tenure to adjust itself during the search than to impose a fixed value.

5 Experimentation and Results

In this section, we present experimental results of the TS algorithm on two sets of real size FAP instances provided by the French National Research Center for Telecommunications. These instances are produced by a generator in such a way that they reflect various situations encountered in real networks or sub-networks of France Telecom. The main goal here consists of finding interference-free assignments ($f(s) = 0$) with a minimal number of frequencies.

Two sets of experiments were carried out. The first one aims to evaluate the performance of the Tabu algorithm by comparing it with a baseline algorithm. The purpose of the second one is to compare the results of the Tabu algorithm with the best known results on the same instances obtained with other methods.

5.1 Tests

Test Set No.1 (Traffic=1)

The first set of FAP instances has the following characteristics.

- *traffic*: $T_i = 1$ (i $\in \{1..N\}$); i.e. each cell is assigned one frequency. Consequently, a co-cell constraint does not exist.
- *adjacent constraint*: $|f_{i,1} - f_{j,1}| \geq M[i,j] = 1$ (i, j $\in \{1..N\}$) for two adjacent cells C_i and C_j; i.e. C_i and C_j must be assigned different frequency values.

It is easy to see that these instances correspond to the classic graph coloring problem. Finding an optimal frequency assignment is equivalent to coloring a graph using only a minimum number (the chromatic number) of colors. For this test set, the chromatic number, noted *Opt* for each instance, is known in advance.

These instances are labeled as *NF.N.d* representing respectively the optimal number of frequencies needed for an interference-free assignment, the number of cells in the network and the density of interference constraints over the network. For example, 8.150.30 defines a network composed of 150 cells with 8 the number of frequencies required in an interference-free assignment and 30% of a total of 150*(150-1)/2 interference constraints.

Test Set No.2 (Traffic=2)

The second set of FAP instances has the following characteristics.

- *traffic*: $T_i = 2$ (i $\in \{1..N\}$); i.e. each cell is assigned two frequencies.
- *co-cell constraint*: $|f_{i,m} - f_{i,n}| \geq M[i,i] = 3$ (i $\in \{1..N\}$); i.e. two frequencies assigned to the same cell must have a minimum distance of 3.
- *adjacent constraint*: $|f_{i,m} - f_{j,n}| \geq M[i,j] \in \{1,2\}$ (i, j $\in \{1..N\}$) if C_i and C_j are adjacent cells; i.e. two frequencies assigned to two adjacent cells must have a minimum distance of 1 or 2 according to the cells.

The instances of this set are labeled as *q.N.d.p*. The first parameter has no special meaning. The second and third parameters have similar meanings to those of the first set. The fourth parameter allows distinguishing between different instances of the same class. For this set of instances, the exact optimal number of frequencies for an interference-free assignment is no longer known due to the way the instances are generated, but it is bounded by a lower bound *LB* given by the generator.

Compared with the first set, these instances are often larger in terms of the number of constraints due to the doubled traffic. Large instances of 300 cells (representing 600 integer variables) contain up to 30,000 interference constraints. As we will see in the next section, these instances are usually harder to solve than those of the first set.

5.2 Performance Evaluation

5.2.1 Baseline Algorithm

In order to evaluate the performance of the Tabu algorithm, we need a reference for comparison. We take the standard steepest descent (SD) algorithm (with re-runs) as a baseline reference for several practical reasons. First, SD is a well known and simple algorithm. Second, SD requires no parameter to tune. Third, from the point of view of implementation, SD and TS may share many important data structures. Forth, the results of SD will give us some indication as to the difficulty of the instances tested.

The SD algorithm consists of repeating the following SD operation:

Steepest descent (SD): At each iteration, choose a best neighbor *s'* $\in$ *V** such that *f(s')* $\leq$ *f(s)* (break ties randomly).

We must point out that the SD algorithm uses the same neighborhood *N(s)*, candidate list *V** and evaluation method as those used by the TS algorithm.

5.2.2 Method of Experimentation and Evaluation Criteria

The TS algorithm is run with initial random configurations. For a fixed number of frequencies, TS is run 10 times, each run being limited to a maximum of 100,000 iterations. Once an interference-free assignment is found with *NF* frequencies, the TS algorithm is called with a new random assignment using *NF-1* frequencies.

The parameter α of tabu tenure ($k = \alpha|V^*|$) was determined according to the following process. Values from 0.1 to 0.5 were tested for each instance with limited iterations (50,000 in this study) and the best one is used in later experiments (100,000 in this study).

In order to make the comparison between TS and SD as fair as possible, we give both algorithms the same number of iterations (100,000). However, while TS may improve its results with more iterations, this is not the case for SD. To avoid this problem, we give SD more re-runs. For each run of TS, SD is run 20 times, each being limited to 5,000 iterations in order to reach 100,000 iterations. The number 5,000 is empirically determined in such a way that it is sufficiently high for SD to reach local optima and move around with some side-walks.

The following three criteria are used to carry out the evaluation.

NF(S) the smallest number of frequencies found for an interference-free assignment, followed in parenthesis by the number of successful runs which find such an interference-free assignment with *NF* frequencies. This criterion reflects the *quality* of a solution.

Iter the number of iterations averaged over successful runs for the smallest number of frequencies found for an interference-free assignment. One iteration corresponds to a move from one solution to one of its neighbors. This criterion reflects the machine-independent *speed* of an algorithm.

T(sec) the average user time[7] in seconds averaged over successful runs for the smallest number of frequencies found. The timing is based on a SPARCstation 5 (75MHz, 32MB RAM).

5.2.3 Experimental Results

Table 1 shows the results of TS and SD on 4 instances of the first set and 6 instances of the second set. These instances are taken from a total of 60 instances and represent some hard problems.

For each instance, we give its name (first column), the optimal number of frequencies (Opt) for the first set or the lower bound (LB) for the second set (2nd column), followed by the results of TS (3rd-7th columns) and SD (8th-11th columns). For instance, the data for *8.150.20* have the following meanings. TS finds *optimal* solutions for each of its 10 runs with an average of 18,923 iterations and 123 seconds per (successful) run. SD finds 8 times (out of 200) an interference-free assignment with 10 frequencies after 347 iterations and 3 seconds while none of its 200 runs manages to find an interference-free assignment with 9 frequencies after 5,000 iterations per run.

From the higher part of Table 1, we see that the 4 instances of the first set are too difficult for SD to solve to optimality. Indeed, SD requires 2 to 8 extra frequencies to find an interference-free assignment. On the contrary, TS solves all the instances to optimality at each run with a maximum of 10 minutes. Following the size and the difficulty of an

[7] Both TS and SD are programmed in C++.

Problems	Opt/LB	TS					SD			
		Runs	α	NF(S)	Iter	T[sec]	runs	NF(S)	Iter	T[sec]
8.150.20	8	10	0.4	8(10)	18923	123	200	10(8)	347	3
8.150.30	8	10	0.2	8(10)	404	3	200	13(14)	484	3
15.300.20	15	10	0.2	15(10)	41484	573	200	17(4)	1646	27
15.300.30	15	10	0.1	15(10)	22429	327	200	23(2)	927	19
8.75.25.1	16	10	0.2	17(10)	34414	274	200	20(2)	170	3
8.75.25.3	16	10	0.3	16(5)	62764	485	200	19(2)	743	9
8.150.15.3	16	10	0.3	18(10)	46425	668	200	22(10)	536	12
8.150.25.6	16	10	0.3	16(3)	56120	906	200	27(10)	986	24
15.300.25.6	30	10	0.2	35(2)	78266	2940	200	49(2)	635	48
15.300.25.9	30	10	0.1	35(2)	61294	2168	200	45(1)	823	65

Table 1: Results of TS and SD, 100,000/5,000 iterations per run for TS/SD

instance, TS does approximately 70 to 150 iterations per second. Since SD and TS gives solutions of highly different quality, it is difficult to compare their solving speeds.

Concerning the instances of the second set (the lower part of Table 1), let us notice that they are more difficult than the previous ones (the two *15.300.25.x* instances seem extremely hard). For these instances, the difference between the results of SD and TS is even larger. Indeed, to find an interference-free assignment, SD requires at least 3 extra frequencies above the lower bound. For the last two instances, this excess number reaches respectively 19 and 15.

TS finds interference-free assignments with the lower bound frequency for 2 of 6 instances within the limit of 100,000 iterations. For the 4 other instances, 1 to 5 extra frequencies above the lower bound are necessary to have an interference-free assignment with success rates ranging from 20% to 100%. As shown in the next section, these results will be improved if more iterations is given to the TS algorithm. For these instances, TS achieves approximately 30 to 120 iterations per second. Finally, TS needs about 5 to 50 minutes to find its solutions for these instances. These computing times are considered to be quite satisfactory since in practice up to 48 hours are allowed to find a frequency assignment. Moreover, minimizing the number of frequencies used is far more important than the solution speed.

5.3 Comparisons with Other Methods

In this section, we contrast the results of the TS algorithm with the best results on the same instances obtained with other methods including algorithms based on Simulated Annealing (SA), Constraint Programming with ILOG-solver (CP) and Graph Coloring (GCA) [11].

The results of TS are those presented above, limited to 100,000 iterations. The results of SA, CP, and GCA are those reported in [1, 2, 19]. Timing for SA and CP (not available for GCA) is based on HP Stations (HP 735, 125 MHz, 128MB RAM) which are considered to be about 2 to 3 times faster than the SPARCstation 5 (75MHz, 32MB RAM) we used. SA and CP are respectively stopped after 2 and 10 hours on HP Stations, equivalent to about 6 and 30 hours on a SPARCstation 5. Table 2 gives a summary of the results of all these methods on the same set of instances. A minus "-" in the table means that the result is not available.

From the data of Table 2, we notice first that TS and SA largely outperform CP and GCA on these instances in terms of the quality of solutions. Indeed, none of these instances is solved to optimality by CP or GCA (with the exception for *8.150.20* by GCA).

Problems	Opt/LB	TS			SA		CP		GCA
		NF(S)	Iter	T[sec]	NF	T[sec]	NF	T[sec]	NF
8.150.20	8	8(10)	18923	123	8	509	9	7200	8
8.150.30	8	8(10)	404	3	8	446	12	10800	15
15.300.20	15	15(10)	41484	573	15	4788	17	3600	20
15.300.30	15	15(10)	22429	327	15	2053	24	36000	27
8.75.25.1	16	17(10)	34414	274	17	1382	20	1000	19
8.75.25.3	16	16(5)	62764	485	17	1744	19	7595	19
8.150.15.3	16	18(10)	46425	668	18	3705	22	375	22
8.150.25.6	16	16(3)	56120	906	17	5981	30	153	29
15.300.25.6	30	35(2)	78266	2940	36	5359	47	380	47
15.300.25.9	30	35(1)	61294	2168	36	6586	45	-	45

Table 2: Comparison of TS with other methods (maximum iterations for TS = 100,000)

Many extra frequencies (up to 17) are needed by CP and GCA to find an interference-free assignment. On the contrary, the results of TS and SA are much better on all the instances. In general, the harder the problem, the bigger the difference: for some instances, there is a difference of more than 12 frequencies.

Let us now examine the results of TS and SA. In terms of the quality of solutions, we see that the TS algorithm outperforms the SA algorithm on 4 instances (in bold) and does as well as SA on the other instances. This is remarkable since reducing even one frequency for such hard problems makes the problem much harder to solve. In terms of computing time, TS is always faster than SA even for solutions of better quality. If we take into account the difference of computing power between the machines used, it should be clear that TS dominates SA.

In order to see how far the TS algorithm can go, another experiment was carried out. In this experiment, we run the algorithm with more iterations (1,000,000 maximum) on 4 instances for which no interference-free assignment was found at 100,000 iterations with lower bound frequencies. Table 3 shows the results of this experiment.

Problems	Runs	TS		
		NF(S)	Iter	T[sec]
8.75.25.1	10	17(10)	46425	441
8.150.15.3	10	16(3)	684200	12478
15.300.25.6	5	32(3)	458764	20729
15.300.25.9	5	34(1)	995667	37410

Table 3: More TS results, maximum iterations = 1,000,000

From Table 3, we see that *8.150.15.3* is solved with 16 frequencies (lower bound). The results of *15.300.25.6* and *15.300.25.9* are improved with a gain of 1 and 3 frequencies respectively. For *8.75.25.1*, no interference-free solution is found with 16 frequencies. To solve the last three instances, a large number of iterations are needed with a running time ranging from 6 to 10 hours.

In summary, we see that the TS algorithm gives the best results for these instances in terms of solution quality and solving speed. This remains true for 50 other instances solved but not reported here. Due to insufficient information about the testing conditions and implementation details of some algorithms, it is difficult to give a really fair comparison. However, contrasting these results seems to give some clear indications about the performance as to these algorithms.

Finally, we point out that in addition to the previous results, the TS algorithm was also applied to real data representing some largest mobile networks used in France to minimize interference for a fixed number of frequencies and to minimize the number of frequencies for interference-free assignments. Due to the confidential nature, we cannot report in details our experiments on these networks. Suffice to say however that the TS algorithm gives very competitive assignments for these real networks compared with other methods.

5.4 Discussions

The TS algorithm presented in this paper offers a significant improvement over the previous best results on the set of tested instances. The high performance of the algorithm comes from the combination of different features. In this section, we study the influence of some of these features.

Conflict-based candidate list strategy

In order to see the influence of this strategy, experiments were carried out using the entire neighborhood *N(s)* and the conflict-based candidate list V^*. We re-run the TS algorithm on some instances with different fixed number of frequencies. Table 4 shows the comparative results on a particular instance (*8.75.25.3*). In the table, each line corresponds to the result of 5 runs and "-" means that no result is found after 100,000 iterations.

NF	N(s)		V^*	
	Iter	T[sec]	Iter	T[sec]
23	237	1.8	86	0.5
22	267	2	104	0.6
21	14596	37	133	0.7
20	-	-	150	1
19	-	-	369	2
18	-	-	1370	10
17	-	-	6481	50
16	-	-	78692	565

Table 4: Influence of candidate list strategy on *8.75.25.3*, maximum iterations = 100,000

We observe from the table that none of the 5 runs (100,000 iterations each) using the entire neighborhood *N(s)* is able to find an interference-free assignment using a number of frequencies smaller than 21. On the contrary, the conflict-based strategy allows the algorithm to find interference-free assignments with only 16 frequencies. Moreover, this strategy is much more efficient in terms of number of iterations and computing time. These remarks remain valid for other tested instances.

Another popular candidate list strategy is random sampling, i.e. only a percentage of the neighborhood is randomly considered at each iteration. Similar tests show that once again the conflict-based strategy largely dominates this sampling technique.

Co-cell constraint handling

In order to see the influence of this technique for the TS algorithm, we re-run our TS algorithm 10 times with the CCH option disabled on the second test set (no co-cell constraint exists in the first test set). Table 5 shows the comparative results of TS with and without CCH.

Problems	LB	Runs	TS with CCH			TS without CCH		
			NF(S)	Iter	T[sec]	NF(S)	Iter	T[sec]
8.75.25.1	16	10	17(10)	34414	274	17(10)	34519	280
8.75.25.3	16	10	16(5)	62764	485	16(4)	41284	319
8.150.15.3	16	10	18(10)	46425	668	18(8)	51555	776
8.150.25.6	16	10	16(3)	56120	906	17(5)	59783	954
15.300.25.6	30	10	35(2)	78266	2940	37(2)	77455	2880
15.300.25.9	30	10	35(1)	61294	2168	36(1)	96722	3511

Table 5: Influence of co-cell constraint handling, maximum iterations = 100,000

From the table, we observe that CCH gives better solutions by reducing the number of frequencies by 1 or 2 for the last 3 instances. Also, to obtain solutions of the same quality, CCH seems to be more robust since the solutions are found more often. More generally, CCH allows to reduce the search space and to accelerate the search process.

Re-generation

Recall that there are two different ways to generate an initial configuration with *NF-1* frequencies when an interference-free assignment is found with *NF* frequencies. One may take a completely new random configuration or re-generate a configuration from the last interference-free assignment (see Section 4.2).

Results represented above are obtained with random initial configurations. It is thus interesting to know if the "re-generation" technique can lead to different results. Limited experiments were carried out for this purpose. We observed that given the same number of iterations, the re-generation often makes the search faster. We also observed that the re-generation may effectively improve the results. In particular, it helps finding interference-free assignments with 30 frequencies (lower bound) for the two largest instances *15.300.25.x*. At the same time, we notice that the improvement is problem-dependent. More work is needed to know to which extent and in which cases the re-generation is beneficial in general.

6 Conclusions and Future Work

In this paper, we have presented a powerful TS algorithm for the frequency assignment problem. The algorithm has some key features including incremental evaluation, co-cell constraint handling, candidate list strategy and dynamic tabu tenure. The algorithm was evaluated against a steepest descent algorithm on various realistic instances. Results of the TS algorithm were also compared with the best results obtained with other algorithms based on simulated annealing, constraint programming and graph coloring. Empirical evidence shows that the TS algorithm outperforms largely other algorithms by producing the best results for the set of tested instances both in terms of quality of solution and solving speed. Consequently, TS should be considered to be a very competitive and promising method for the frequency assignment problem.

The FAP instances solved in this work may be considered to be large for today's networks. However, mobile radio networks grow extremely fast and will reach easily 1,000 to 2,000 cells (several thousands of variables) in the near future. Therefore, very powerful optimization methods are needed to deal with such networks. Parallel and distributed TS techniques may be considered since they are probably among the most promising methods for tackling large and challenging combinatorial problems [5].

Acknowledgments

We would like to thank gratefully Dr. A. Caminada from CNET - France Telecom for his assistance. We also thank the referees of the paper for their useful comments.

References

[1] A. Caminada, "Résolution du Problème de l'Affectation des Fréquences Par Programmation Par Contraintes", Technical Report FT.CNET/BEL/POH/CDI/71-95/CA, CNET, Belfort, 1995.

[2] A. Caminada, Some Results of Constraint Programming for Frequency Assignment, Personal Communication, 1996.

[3] CALMAR Symposium on Combinatorial Algorithms for Military Applications, Scheveningen, Holland, November 1995.

[4] D.J. Castelino, S. Hurley and N.M. Stephens, "A Tabu Search Algorithm for Frequency Assignment", Annals of Operations Research, Vol.63, pp301-319, 1996.

[5] T. G. Crainic, M. Toulouse and M. Gendreau, "Towards a Taxonomy of Parallel Tabu Search Heuristics", ORSA Journal of Computing, (to appear).

[6] W. Crompton, S. Hurley and N.M. Stephens, "A Parallel Genetic Algorithm for Frequency Assignment Problems", Proc. of IMACS SPRANN'94, pp81-84, 1994.

[7] R. Dorne and J.K. Hao, "Constraint Handling in Evolutionary Search: A Case Study on Frequency Assignment", 4th Int. Conf. on Parallel Problem Solving from Nature, Lecture Notes in Computer Science 1141, pp801-810, Springer-Verlag, 1996.

[8] M. Duque-Antòn, D. Kunz and B. Rüber, "Channel Assignment for Cellular Radio Using Simulated Annealing", IEEE Trans. on Vehicular Technology, Vol.42, pp14-21, 1993.

[9] C. Fleurent and J.A. Ferland, "Genetic and Hybrid Algorithms for Graph Coloring", in G. Laporte, I. H. Osman, and P. L. Hammer (Eds.), Annals of Operations Research, "Metaheuristics in Combinatorial Optimization", Vol.63, 1996.

[10] N. Funabiki and Y. Takefuji, "A Neural Network Parallel Algorithm for Channel Assignment Problems in Cellular Radio Network", IEEE Trans. on Vehicular Technology, Vol.41, pp430-437, 1992.

[11] A. Gamst, "A Ressource Allocation Technique for FDMA Systems", Alfa Frequenza, Vol.57, No.2, pp89-96, 1988.

[12] F. Glover and M. Laguna, "Tabu Search", Kluwer Academic Publishers, Boston, USA, 1997.

[13] W.K. Hale, "Frequency Assignment: Theory and Application", Proceedings of the IEEE, Vol.68, No.12, pp1498-1573, 1980.

[14] J.K. Hao and R. Dorne, "Study of Genetic Search for the Frequency Assignment Problem", Artificial Evolution, Lecture Notes in Computer Science 1063, pp333-344, Springer-Verlag, 1996.

[15] J.K. Hao and L. Perrier, "Tabu Search for the Frequency Assignment Problem in Cellelar Radio Networks", French Workshop on Practical Solving of NP-Complete Problems, Dijon, France, March, 1996.

[16] A. Hertz and D. de Werra, "Using Tabu Search Techniques for Graph Coloring". Computing Vol.39, pp345-351, 1987.

[17] A. Kapsalis, V.J. Rayward-Smith and G.D. Smith, "Using Genetic Algorithms to Solve the Radio Link Frequency Assignment Problem", Proc. of Intl. Conf. on ANN and GAs, pp37-40, Alès, France, 1995.

[18] D. Kunz, "Channel Assignment for Cellular Radio Using Neural Networks", IEEE Trans. on Vehicular Technology, Vol.40, pp188-193, 1991.

[19] A. Ortega, J.M. Raibaud, M. Karray, M. Marzoug, and A. Caminada, "Algorithmes de Coloration des Graphes et d'Affectation des Fréquences: Approches Géométrique et Analytiques, Heuristiques, Recuit Simulé, Programmation par Contraintes, Algorithmes Génétiques", Technical Report NT/PAB/SRM/RRM/4353, CNET, Paris, August 1995.

[20] F.S. Roberts, "T-colorings of Graphs: Recent Results and Open Problems", Discrete Mathematics, Vol.93, pp229-245, 1991.