Filtrage par arc-consistance et recherche tabou pour l'allocation de fréquences avec polarisation

Michel Vasquez, Audrey Dupont
Centre LGI2P
Ecole des Mines d'Alès
Site EERIE, Nîmes
email: {vasquez,dupont}@site-eerie.ema.fr

Résumé

Cet article présente deux versions d'un algorithme développé dans le cadre de la résolution approchée du problème *FAPP* (**F***requency* **A***ssignment with* **P***olarization* **P***roblem*). L'une des versions utilise la recherche tabou seule, l'autre hybride cette métaheuristique avec un filtrage par consistance d'arc. Nous comparerons le comportement de ces deux versions, sur 30 instances avec plusieurs exécutions, ainsi que la qualité des résultats obtenus.

Mots-clé : Affectation de fréquences, programmation par contraintes, recherche locale tabou.

1 Introduction

Le problème d'allocation de fréquences avec polarisation (*FAPP*) consiste à assigner, à chaque trajet reliant deux tours hertziennes, une fréquence et une polarisation satisfiant toutes les contraintes impératives et le plus grand nombre de contraintes électro-magnétiques, afin de limiter les interférences. L'affectation de fréquences, un des problèmes industriels de références en optimisation combinatoire, est un problème NP-difficile et s'implémente comme un *Max–CSP*.

La méthode tabou que nous avons développée pour résoudre le *FAPP*, est une méthode de recherche locale incluant un voisinage efficace qui travaille sur des instanciations partielles et un traitement actif des contraintes durant les mouvements. Cette partie de l'algorithme est fortement inspirée des travaux précédents [12], qui ont fourni de très bons résultats sur d'autres CSP (*Constraint Satisfaction Problem*).

D'autre part, la grande taille des *benchmarks* ainsi que la limitation du temps cpu, rendent nécessaire la réduction l'espace de recherche. La structure spécifique du réseau de contraintes du *FAPP* permet aisément d'appliquer des techniques locales de filtrage. L'idée est venue naturellement de combiner, à cet algorithme de recherche tabou, une procédure de filtrage par arc-consistance (AC) qui est la consistance locale la plus couramment utilisée. De ce fait, nous situerons notre discussion sur la comparaison des différents comportements des deux versions de l'algorithme :

- la première étant une recherche tabou seule,
- la seconde étant une hybridation de tabou avec la méthode de filtrage AC.

Ce papier se décompose en trois parties : la première décrit le problème de façon générale, la deuxième détaille l'implémentation des deux versions de l'algorithme, et enfin, la troisième commente les résultats trouvés dans le but d'évaluer les gains obtenus par combinaison d'un outil de programmation par contraintes à une heuristique de recherche locale.

2 Définition du problème

Le problème *FAPP* a été largement décrit dans [4]. Nous rappelons, ici, les éléments nécessaires à la bonne compréhension de ce papier.

2.1 Description physique

Le *FAPP* est un problème d'allocation de fréquences dans un réseau hertzien de télécommunications. Le réseau est composé d'un ensemble de sites regroupant l'équipement de transmission, dont les antennes reliant les émetteurs et les récepteurs. Une connection hertzienne entre deux sites géographiques se compose d'un ou plusieurs trajets. Un trajet est un lien radio-électrique continu, établi entre les antennes de sites distincts, auquel on affecte une fréquence et une polarisation. Dans la suite, T dénotera l'ensemble des trajets d'une instance du problème.

Une ressource de fréquence est définie par la paire (fréquence, polarisation) dans laquelle les composants représentent respectivement la fréquence effective du signal transmis et la polarisation d'ondes. Pour simplifier le modèle, nous considérerons la polarisation comme étant une variable binaire, dont les valeurs sont -1 et 1 .
Pour chaque trajet $i \in T$, on définit un ensemble de ressources disponibles, appelé domaine du trajet, par le produit cartésien des domaines de fréquences et des domaines de polarisation. Deux trajets peuvent avoir des domaines identiques. L'allocation de fréquences consiste donc à obtenir pour chaque trajet une paire (fréquence, polarisation) satisfaisant un certain nombre de contraintes de compatibilité radio-électrique, auxquelles on rajoute plusieurs contraintes de couplage :

1. égalité ou inégalité des fréquences des trajets : $f_i = f_j$ ou $f_i \neq f_j$;
2. distance entre les fréquences : $|f_i - f_j| = \varepsilon_{ij}$ ou $|f_i - f_j| \neq \varepsilon_{ij}$;
3. égalité ou inégalité des polarisations des trajets : $p_i = p_j$ ou $p_i \neq p_j$;
4. écart minimal entre les fréquences : $|f_i - f_j| \geq \begin{cases} \gamma_{ij} & \text{si} \quad p_i = p_j \\ \delta_{ij} & \text{si} \quad p_i \neq p_j \end{cases}$.

Dans la quatrième contrainte (CEM pour contrainte d'écart minimal), la distance requise entre les fréquences dépend de leur polarisation. La contrainte est moins forte si les polarisations sont différentes $\delta_{ij} \leq \gamma_{ij}$. La polarisation autorise donc une *relaxation* de la contrainte CEM. Malheureusement, pour la plupart des problèmes, cette relaxation ne permet pas toujours de trouver des solutions réalisables. Pour contrôler le niveau de dégradation des contraintes CEM, il a été introduit un niveau de repli k. Plus précisément, soient:

- CI : contraintes fortes ou impératives (1-3 ci-dessus);
- CEM_k : contraintes du type 4 pour lesquelles une relaxation progressive est autorisée. Le niveau 0 indique qu'il n'y a pas de relaxation; passer du niveau k au niveau $k+1$ implique la relaxation de certaines ou de toutes les contraintes CEM, le niveau maximum étant 10, on a :

$$|f_i - f_j| \geq \begin{cases} \gamma_{ij}^0 \geq \gamma_{ij}^1 \geq \ldots \geq \gamma_{ij}^k \geq \ldots \geq \gamma_{ij}^{10} & \text{si} \quad p_i = p_j \\ \delta_{ij}^0 \geq \delta_{ij}^1 \geq \ldots \geq \delta_{ij}^k \geq \ldots \geq \delta_{ij}^{10} & \text{si} \quad p_i \neq p_j \end{cases}$$

Ainsi, une solution réalisable au niveau k est l'assignement d'une paire (fréquence, polarisation) à chaque trajet qui satisfait toutes les contraintes impératives (CI) et toutes les contraintes d'écart minimal au niveau de repli k (CEM_k). Un tel problème est appelé k–faisable. Dans le cas où il n'existe pas de solution réalisable, la nouvelle fonction objectif devient, dans l'ordre des priorités :

1. chercher pour k^*, le plus bas niveau de relaxation pour lequel une solution k^*–faisable existe;
2. minimiser le nombre de contraintes CEM non satisfaites au niveau $k^* - 1$;
3. minimiser le nombre de contraintes CEM non satisfaites aux niveaux inférieurs à $k^* - 1$.

Cette fonction objectif, caractérisée par ces trois composantes, définit ainsi un problème d'optimisation hiérarchique.

2.2 Le *FAPP* vu comme un *Max–CSP*

En posant les trajets comme variables, les fréquences et les polarisations comme valeurs de domaine pour ces variables et les deux ensembles de contraintes CI et CEM, le *FAPP* peut-être modélisé comme un problème visant à maximiser le nombre de contraintes satisfaites (*Max–CSP*). Plus précisément, nous définissons :

- T l'ensemble des trajets;
- $\{-1\}$, $\{1\}$ et $\{-1,1\}$, les trois domaines de polarisation possibles;
- pour chaque $i \in T$, on associe : $\begin{cases} f_i \in F_i \\ p_i \in P_i \end{cases}$ où F_i et P_i indiquent respectivement les domaines de fréquence et de polarisation de i;
- CI l'ensemble des contraintes :
  $|f_i - f_j| = \varepsilon_{ij}$, $|f_i - f_j| \neq \varepsilon_{ij}$, $f_i = f_j$, $f_i \neq f_j$, $p_i = p_j$ et $p_i \neq p_j$;
- CEM_k, pour $0 \leq k \leq 10$, l'ensemble des contraintes :
  $|f_i - f_j| \geq \frac{|p_i + p_j|}{2} \gamma_{ij}^{(k)} + \frac{|p_i - p_j|}{2} \delta_{ij}^{(k)}$;
- $V^{(k)}$ l'ensemble des contraintes insatisfaites de CEM_k.

Tous les problèmes sont 11–faisables, c'est à dire que le sous-réseau composé uniquement des contraintes CI est satisfiable.
La fonction objectif à minimiser se décompose alors en trois parties, par ordre de priorité décroissante :

1. minimiser k;
2. à k égal, minimiser $V^{(k-1)}$, le nombre de contraintes CEM violées à k-1;
3. à k égal et $V^{(k-1)}$ égaux, minimiser $\sum_{i=0}^{k-2} V^{(i)}$, la somme des contraintes CEM violées des niveaux 0 à k-2.

Afin de simplifier les notations suivantes, nous définissons :

- $C_k = CI \cup CEM_k$ l'ensemble de toutes les contraintes au niveau k;
- $CI(ij) \subset CI$ les contraintes impératives entre les trajets i et j;
- $CEM_k(ij) \subset CEM_k$ les contraintes d'écart entre les fréquences des trajets i et j.

3 Approche générale pour résoudre le *FAPP*

Parmi les algorithmes utilisés pour résoudre les problèmes d'affectation de fréquences [1, 5, 8], seuls ceux basés sur des méthodes approchées fournissent de bonnes solutions en un temps raisonnable, sur des instances de tailles réelles. L'algorithme décrit ici, en fait parti. Un mécanisme de recherche locale tabou est renforcé par une technique de filtrage par consistance d'arc, qui réduit la taille de l'espace de recherche et augmente l'efficacité globale de l'exploration du voisinage. En effet :

- soit $n = |T|$ le nombre de trajets d'une instance du *FAPP*;
- soit S l'espace de recherche non-contraint : S inclut tous les vecteurs de configuration $s = (f_1, p_1, \cdots, f_n, p_n)$ tels que $f_i \in F_i$ et $p_i \in P_i$ même si les contraintes C_k sont insatisfaites.

La taille initiale de S est égale à $\prod_1^n (|F_i| \times |P_i|)$, ce qui est une valeur énorme même pour les plus petites instances du *FAPP* ($200 \leq n \leq 3000$ et $19 \leq |F_i| \leq 500$). De par la nature combinatoire du problème, un pré-traitement des données par une méthode de filtrage peut sembler dénué d'intérêt, puisqu'il réduit un ensemble exponentiel en un ensemble lui aussi exponentiel, à peine plus petit. Cependant, la réduction de la taille du domaine apporte deux avantages majeurs. Le premier est la diminution de la complexité de calcul lors de l'évaluation du voisinage. Le second est qu'il évite l'exploration de mauvaises zones de l'espace de recherche dès lors qu'un *nogood* est détecté. Ainsi la réduction de la taille de l'espace de recherche permet d'explorer plus de configurations et de meilleures qualités.

Une autre caractéristique importante de cette approche, issue du même désir de diminuer d'autant que possible la complexité de calcul, est de ne prendre en compte que la première composante de la fonction objectif f (voir fin du paragraphe 2.2); ce qui signifie que le but principal du processus d'optimisation est de trouver une configuration qui minimise le niveau k de faisabilité. De ce fait, la minimisation du k se traduit simplement par une série de problèmes de décision pour lesquels l'objectif est de trouver une configuration k-faisable de $k = k_0 = 11$ à $k = k^* \leq k_0$. La recherche d'une valeur k est lancée seulement si une configuration $(k+1)$–faisable a été trouvée.

En résumé, si AC est la fonction qui filtre le domaine de valeurs et TABOU celle qui explore l'espace de recherche en utilisant la métaheuristique tabou afin de fournir une solution k–faisable, nous avons le schéma général suivant :

Algorithme 1 : AC-TABOU

début
  $k \leftarrow 11$
  tant que $AC(k) = Vrai$ **faire**
    si TABOU$(k) = Vrai$ **alors**
      $k \leftarrow k - 1$
fin

C'est une procédure itérative très simple. Lorsque $AC(k)$ ou TABOU(k) échoue, AC-TABOU retourne la meilleure solution k^*–faisable trouvée, avec $k^* = k + 1$. Il ne reste qu'à spécifier ce que $AC(k) = Vrai$ et TABOU$(k) = Vrai$ signifient.

3.1 Filtrer l'espace de recherche avec AC

L'arc-consistance a été largement étudiée en programmation par contraintes [2, 3, 9, 10]. Elle consiste à éliminer les valeurs des domaines qui n'ont pas de supports, c'est à dire celles qui ne peuvent conduire à une solution faisable en utilisant des considérations locales entre les variables impliquées dans une même contrainte.

Comme nous l'avons vu plus tôt, les contraintes du *FAPP* sont binaires : chacune d'entre elles définie une relation entre deux variables (*trajets*) du problème. L'ensemble de ces relations dessine donc un graphe de contraintes où les noeuds correspondent aux trajets et les arêtes aux contraintes de CI et CEM_k. Dans ce cas il est facile de vérifier pour chaque trajet $i, 1 \leq i \leq n$, et pour chaque paire (f_i,p_i) (*fréquence,polarisation*) si, pour chaque voisin j de i, il y a une paire de valeurs (f_j,p_j) telle que $CI(ij)$ et $CEM_k(ij)$ soient satisfaites. Ce processus d'élimination (*éliminer*(F_i,P_i,k)) est répété jusqu'à ce qu'il n'y ait plus de changements dans les domaines des variables, dans ce cas *éliminer* retourne $Faux$ sinon il retourne $Vrai$.

Algorithme 2 : AC(k)

début
  change $\leftarrow Vrai$
  tant que *change* **faire**
    change $\leftarrow Faux$
    pour $i \in [1,n]$ **faire**
      si *éliminer*(F_i,P_i,k) **alors**
        si $F_i = \emptyset$ *ou* $P_i = \emptyset$ **alors**
          retourner $Faux$
        change $\leftarrow Vrai$
fin

Pour des informations plus détaillées sur cet algorithme, le lecteur pourra se référer à [2, 3, 9]. Les principales caractéristiques de cet algorithme sont :

- il gère une liste des variables, dont les voisins dans le réseau de contraintes ont changé, afin qu'ils soient contrôlés à la prochaine boucle *élimination* (comme AC3, décrit dans les références données ci-dessus);
- pour chaque valeur de $F_i \times P_i$, et pour chaque contrainte $C_k(ij)$, la fonction *éliminer*(F_i,P_i,k) passe au i suivant, dès qu'elle a trouvé un support (f_j,p_j) dans $F_j \times P_j$ pour la valeur (f_i,p_i).

3.2 Explorer l'espace de recherche avec TABOU

La recherche tabou est une métaheuristique qui guide la recherche locale à partir d'une liste de configurations interdites, conçue pour éviter les cycles ou les phases d'intensification/diversification, utiles pour une meilleure exploration de l'espace combinatoire [7].

Plusieurs points importants sont à considérer dans l'optique d'obtenir une *bonne* méthode tabou : l'espace de recherche effectif et, par conséquent, la définition de configurations s visitées par le processus d'optimisation, la structure de voisinage $N(s)$, l'heuristique de mouvement, etc.

Comme nous l'avons vu dans l'introduction (section 1) l'algorithme TABOU développé pour le *FAPP* est fortement inspiré de travaux précédents [12], dont les principales caractéristiques sont :

- un traitement actif des contraintes binaires : CI et CEM_k restent satisfaites pendant le processus de recherche ;
- un *voisinage original* $N(s)$ définit par un mouvement multi-attributs, qui consiste à allouer une paire (fréquence,polarisation) au trajet i, puis de désallouer tous les trajets qui sont en conflit avec la nouvelle affectation (ensemble de trajet $\{j\}$) par rapport aux contraintes $CI(ij)$ et $CEM_k(ij)$;
- une gestion de la liste tabou basée sur le nombre de mouvements.

A la définition précédente de S (cf. section 3) nous ajoutons :

- $X_k \in S$ le sous-ensemble des configurations k–faisable ;
- Y_k l'ensemble des configurations *partielles* s pour lesquelles le vecteur des variables allouées est k–faisable : $2^{X_k} \subseteq Y_k$;
- $|s|$, $\forall s \in Y_k$, le nombre de trajets alloués dans s.

Il est évident que si $s \in Y_k$ vérifie $|s| = n$ alors $s \in X_k$. Si nous notons $\Delta(s,s') = |s'| - |s|$ la fonction *heuristique* pour évaluer les mouvements potentiels de $s \in Y_k$ à $s' \in N(s)$ (noté $mvt(s,s')$), nous pouvons donner l'algorithme général de recherche TABOU suivant.

Algorithme 3 : TABOU(k)

début
$s \leftarrow \vec{0}$ %*i.e.* aucun trajet n'est alloué : $|s| = 0$
$s^* \leftarrow s$ % la meilleure configuration trouvée auparavant
tant que *critère-arrêt* = *Faux* **faire**
  $s^{max} \leftarrow \vec{0}$ % le meilleur voisin de s
  pour $s' \in N(s)$ **faire**
    si $\Delta(s,s') > 0 \vee mvt(s,s')$ *n'est pas tabou* **alors**
      si $\Delta(s,s') > \Delta(s,s^{max})$ **alors**
        $s^{max} \leftarrow s'$
  si $s^{max} \neq \vec{0}$ **alors**
    $s \leftarrow s^{max}$ % $mvt(s,s^{max})$
    si $|s| > |s^*|$ **alors**
      $s^* \leftarrow s$
      si $|s| = n$ **alors**
        retourner $Vrai$
    mettre à jour la liste tabou
  sinon
    critère-arrêt$\leftarrow Vrai$
retourner $Faux$
fin

Rappelons que le temps de calcul est limité, ce qui implique que le *critère-arrêt* peut être mis à *Vrai* par une interruption d'unité centrale de traitement d'horloge. L'algorithme choisit l'allocation de trajets qui désinstancie le moins de trajets déjà assignés.

La liste tabou est contrôlée par une table *tabou* de $\sum_1^n(|F_i| \times |P_i|)$ entrées qui stocke la valeur $iter + freq[i]$ où $iter$ est le compteur de mouvement et $freq$ est une autre table à $\sum_1^n(|F_i| \times |P_i|)$ entrées. A chaque mouvement $mvt(s,s')$ qui consiste à allouer à un trajet i, une fréquence et une polarisation telles que la paire (f_i,p_i) soit la $k^{ième}$ valeur du domaine du trajet i, l'entrée correspondante $freq[i,k]$ est incrémentée.
A chaque fois qu'un tel mouvement n'améliore pas $|s^*|$, les entrées $tabou[j,k_j]$ de toutes les variables j désinstanciées sont alors mises à $iter + freq[j,k_j]$. Un mouvement est tabou lorsque $tabou[j,k_j] \geq iter$. Ceci permet donc de désallouer certains trajets de s, avec l'assurance qu'ils ne seront pas ré-allouer avec la même paire (f_i,p_i) dans les prochains mouvements.

D'autre part, l'évaluation Δ du voisinage est effectué très rapidement en utilisant des techniques incrémentales efficaces, qui sont maintenant bien connues [6, 12].

3.3 Diversification

Le but de la diversification est de permettre à TABOU de s'échapper des zones attractives (par exemple, les minima locaux) de l'espace de recherche.

Pour ce faire, nous introduisons des pénalités dans l'heuristique de mouvement. Plus précisément, à chaque fois que l'on rencontre une configuration s telle que $\forall s' \in N(s)$, $|s'| \leq |s|$ (s a plus de trajets alloués que toutes ses configurations voisines), nous ajoutons une pénalité à toutes les paires (*variable, valeur affectée*) des trajets déjà alloués ayant un voisin *dans le graphe de contraintes* qui n'est pas alloué. En effet, ces trajets et leurs valeurs assignées constituent un *nogood*, qu'on stigmatise en incrémentant la pénalité. Cette valeur de pénalité est incluse dans la fonction heuristique Δ utilisée pour choisir les mouvements pendant la phase de diversification.

4 Expérimentations

Les algorithmes sont programmés en C; les tests ont été effectués sur un PC (Pentium III, 600 MHz) sous Windows NT.

La présentation des résultats est divisée en trois parties : la première montre l'impact positif de l'étape de filtrage sur la taille de l'espace de recherche; la deuxième décrit les améliorations qualitatives des résultats obtenus par l'approche hybride; et la troisième termine par l'étude comparative des résultats produits par les deux versions (*pure* et hybride) de l'algorithme de recherche TABOU.

4.1 Réduction du domaine de fréquences

Ce premier tableau donne quelques caractéristiques des instances du *FAPP* : leurs tailles : jusqu'à **3000** variables (*i.e.* trajets), leurs domaines : jusqu'à **2087947** valeurs et le nombre de contraintes : jusqu'à **67898** contraintes binaires.

fapp	*IDS*	*FDS*	$\Delta\%$	k_0	sec.	fapp	*IDS*	*FDS*	$\Delta\%$	k_0	sec.
01_0200	26963	12712	52.85	2	0	16_0260	47293	46622	1.42	10	0
02_0250	36618	14759	59.69	1	2	17_0300	64034	918	98.57	3	0
03_0300	53536	28212	47.30	6	1	18_0350	73016	1089	98.51	7	0
04_0300	61762	21962	64.44	0	4	19_0350	201074	3414	98.30	5	6
05_0350	79311	54177	31.69	7	2	20_0420	87077	1886	97.83	9	0
06_0500	108024	53034	50.91	4	6	21_0500	113594	7745	93.18	3	1
07_0600	109658	69952	36.21	8	2	22_1750	813037	10656	98.69	6	25
08_0700	134020	81933	38.87	4	5	23_1800	455735	3265	99.28	8	9
09_0800	121824	52948	56.54	2	6	24_2000	567396	8328	98.53	6	4
10_0900	197665	122050	38.25	5	8	25_2230	610084	18867	96.91	2	6
11_1000	294634	152727	48.16	7	15	26_2300	635123	14217	97.76	6	4
12_1500	436967	164613	62.33	1	70	27_2550	588188	93768	84.06	4	8
13_2000	320494	144873	54.80	2	21	28_2800	2087947	63597	96.95	2	66
14_2500	774322	320458	58.61	3	92	29_2900	1477634	6435	99.56	5	23
15_3000	515606	306127	40.63	4	24	30_3000	1942250	80703	95.84	6	103

TAB. 1 – *Réduction de l'espace de recherche*

Le tableau 1 montre la taille initiale du domaine ($IDS = \sum_1^n |D_i^0|$) et la taille finale (*ou filtrée*) ($FDS = \sum_1^n |D_i^f|$) résultant du calcul du filtrage arc-consistant. Nous obtenons de très bon taux ($\Delta\%= (IDS - FDS)/IDS$) pour une grande partie des instances, notamment pour l'instance 29_2900 (99.56) qui est un des plus gros problème.

Les instances sont numérotées de 01 à 30; la deuxième partie de leur nom correspond au nombre de trajets : ainsi, la $29^{ième}$ instance est un problème de 2900 variables. La colonne k_0 indique le niveau d'infaisabilité le plus élevé rencontré par le filtrage AC. La colonne *sec.* donne la durée de calcul en secondes du processus complet de filtrage itératif. Considérant encore une fois la $29^{ième}$ instance, cela signifie que la méthode AC a mis 23 secondes pour effectuer 7 passages (de $k = 11$ à $k = 5$). Par conséquent, moins de deux minutes sont nécessaires pour réduire les valeurs de domaine des plus grandes instances de ce benchmark.

Mis à part l'instance 16_0260, ces jeux d'essai peuvent être classés en deux sous-ensembles; le premier (*de 01 à 15*) contient les instances pour lesquelles la structure du réseau de contraintes permet de supprimer seulement la moitié des valeurs possibles par des considérations locales, tandis que celle du deuxième sous-ensemble est assez forte pour permettre de supprimer environ 90% des valeurs et donc de réduire considérablement la taille de l'espace de recherche.

4.2 Résultats de l'approche hybride

Nous commençons cette présentation en donnant les résultats obtenus par l'approche hybride après 1 heure de calcul et seulement une exécution (avec la valeur de germe 0 pour le `srand()`, fonction standard de C).

fapp	k	v_{k-1}	$\sum v_{k-2}$	t_1	t_2	fapp	k	v_{k-1}	$\sum v_{k-2}$	t_1	t_2
01_0200	4	4	203	2	1839	16_0260	$\underline{11}$	382	3864	0	2976
02_0250	$\underline{2}$	16	173	19	19	17_0300	$\underline{4}$	4	36	1	1
03_0300	$\underline{7}$	28	835	30	34	18_0350	$\underline{8}$	4	55	1	1
04_0300	$\underline{1}$	97	0	26	2595	19_0350	$\underline{6}$	3	66	16	16
05_0350	11	1	1836	1	3571	20_0420	$\underline{10}$	6	133	2	2
06_0500	$\underline{5}$	30	764	232	233	21_0500	$\underline{4}$	2	12	2	2
07_0600	$\underline{9}$	85	3039	107	433	22_1750	$\underline{7}$	22	383	76	76
08_0700	$\underline{5}$	67	1131	21	35	23_1800	$\underline{9}$	16	189	31	31
09_0800	$\underline{3}$	71	707	1789	1796	24_2000	$\underline{7}$	9	91	17	17
10_0900	$\underline{6}$	74	1877	174	372	25_2230	$\underline{3}$	7	33	19	19
11_1000	$\underline{8}$	105	4278	63	139	26_2300	$\underline{7}$	8	75	18	18
12_1500	4	87	2138	2873	2873	27_2550	$\underline{5}$	9	46	21	21
13_2000	5	126	3469	1020	1020	28_2800	$\underline{3}$	38	125	165	203
14_2500	6	211	6302	1875	2950	29_2900	$\underline{6}$	25	280	86	86
15_3000	$\underline{5}$	198	4770	2534	2807	30_3000	$\underline{7}$	36	798	268	275

TAB. 2 – AC-TABOU *résultats après 1 heure de calcul*

Pour chaque instance, le tableau 2 spécifie :
- k : le plus bas niveau de faisabilité d'une solution. Cette valeur est soulignée quand AC prouve que c'est l'optimum;
- v_{k-1} : le nombre de contraintes de CEM violées au niveau $k-1$;
- $\sum v_{k-2}$: la somme des contraintes de CEM violées en dessous du niveau $k-1$;
- t_1 : le temps (en secondes) pour trouver la meilleure valeur de k;
- t_2 : le temps cpu utilisé pour obtenir la meilleure configuration (en tenant compte des trois composantes de la fonction objectif).

Une première qualité de cet algorithme AC-TABOU est qu'il prouve l'optimalité du niveau de repli sur la majorité des instances. Il n'est pas surprenant que les instances pour lesquelles AC-TABOU ne prouve pas l'optimum sont celles qui appartiennent au premier sous-ensemble.

Maintenant regardons ce qui se produit si nous donnons plus de temps cpu à l'algorithme hybride AC-TABOU.

fapp	k	v_{k-1}	$\sum v_{k-2}$	t_1	t_2
01_0200	4	4	139	2	10194
05_0350	11	1	1683	1	5446
12_1500	$\underline{2}$	57	1263	5384	5388
13_2000	$\underline{3}$	165	2253	7345	7686
14_2500	5	168	4700	10450	10450
16_0260	$\underline{11}$	358	3656	0	8543

TAB. 3 – *Résultats améliorés après 3 heures de calcul*

Le tableau 3 ne montre que les instances sur lesquelles les 3 heures de calcul ont permis de trouver des meilleures solutions. Or, ces améliorations ne concernent qu'un cinquième des instances et principalement celles appartenant au premier sous-ensemble du benchmark. Cette augmentation de temps de calcul a quand même mis en évidence deux nouvelles valeurs optimales du niveau de faisabilité (pour les instances 12_1500 et 13_2000).

4.3 TABOU **versus** AC-TABOU

Dans cette dernière partie, nous allons comparer les comportements des deux algorithmes : recherche simple TABOU et hybride AC-TABOU. Les algorithmes sont exécuté 8 fois avec des valeurs de germe de 0 à 7 pour la fonction `srand()`, et un temps de calcul d'une heure par exécution.

Le tableau 4 concerne le niveau de k-faisabilité et donne la meilleure solution (k_{min}) et la pire (k_{max}) trouvées par la méthode *pure* TABOU et la méthode hybride AC-TABOU. Il est à noter que les résultats des instances 12_1500, 13_2000 et 14_2500 ne contredisent pas ceux du tableau 3 : cela signifie simplement que AC-TABOU trouve une meilleure solution en 3 heures avec une valeur de germe de 0, qu'en 8 $\times$ 1 heures avec différentes valeurs de germe. Plus généralement, le comportement de AC-TABOU est plus stable que TABOU seul.

Finalement, le tableau 5 donne les meilleures solutions trouvées par les deux versions de l'algorithme de recherche TABOU (*pure* et *hybride*), soit en une exécution de 3 heures, soit en huit d'une heure.

	AC-Tabou		Tabou			AC-Tabou		Tabou	
`fapp`	k_{min}	k_{max}	k_{min}	k_{max}	`fapp`	k_{min}	k_{max}	k_{min}	k_{max}
01_0200	4	4	4	4	16_0260	11	11	11	11
02_0250	2	2	2	3	17_0300	4	4	4	4
03_0300	7	7	7	7	18_0350	8	8	8	8
04_0300	1	1	1	11	19_0350	6	6	6	9
05_0350	11	11	11	11	20_0420	10	10	10	10
06_0500	5	5	5	6	21_0500	4	4	4	5
07_0600	9	10	9	10	22_1750	7	7	7	11
08_0700	5	6	5	6	23_1800	9	9	9	11
09_0800	3	4	3	8	24_2000	7	7	7	11
10_0900	6	7	6	11	25_2230	3	3	10	11
11_1000	8	9	8	11	26_2300	7	7	7	10
12_1500	4	7	6	8	27_2550	5	5	11	11
13_2000	5	6	6	6	28_2800	3	3	9	11
14_2500	6	6	7	11	29_2900	6	6	8	12
15_3000	5	6	7	11	30_3000	7	7	11	11

Tab. 4 – *Meilleurs et plus mauvais résultats pour le niveau de faisabilité*

`fapp`	k	v_{k-1}	$\sum v_{k-2}$	t_1	t_2	`fapp`	k	v_{k-1}	$\sum v_{k-2}$	t_1	t_2
01_0200	4	4	139	2	10194	16_0260	11	**5**	**56**	0	1526
02_0250	2	**9**	**186**	28	606	17_0300	4	4	**34**	1	16
03_0300	7	**14**	**715**	35	2789	18_0350	8	4	55	1	1
04_0300	1	97	0	20	1982	19_0350	6	2	51	696	1962
05_0350	11	1	1683	1	5446	20_0420	10	6	118	1	20
06_0500	5	**22**	**697**	2877	3370	21_0500	4	2	10	176	2314
07_0600	9	**48**	**2507**	11	2954	22_1750	7	15	291	1694	2861
08_0700	5	**34**	**929**	68	1285	23_1800	9	16	189	23	454
09_0800	3	**46**	**652**	96	2270	24_2000	7	**6**	**71**	25	741
10_0900	6	**34**	**1545**	58	2479	25_2230	3	7	33	14	14
11_1000	8	**25**	**3334**	188	734	26_2300	7	8	**69**	1623	2441
12_1500	2	57	1263	5384	5388	27_2550	5	9	46	21	21
13_2000	3	165	2253	7345	7686	28_2800	3	38	125	165	185
14_2500	5	168	4700	10450	10450	29_2900	6	25	280	66	66
15_3000	5	187	4774	2060	2455	30_3000	7	36	798	205	209

Tab. 5 – *Meilleurs résultats*

La plupart des valeurs de k ne sont plus soulignées, ce qui signifie que la majorité des meilleurs résultats (concernant les autres composantes v_{k-1} et $\sum v_{k-2}$ de la fonction objectif) ont été atteints par l'algorithme de recherche Tabou *pure*. En revanche, les améliorations des valeurs des tableaux 2 et 3 (résultats en **gras** dans ce tableau) concernent essentiellement les instances du premier sous-ensemble de problèmes. Nous expliquons ce *bon* comportement de l'algorithme Tabou par l'originalité de son voisinage qui tient compte de la consistance locale.

5 Conclusion

Dans ce papier, nous avons présenté une approche hybride pour résoudre le problème d'affectation de fréquences avec polarisation. L'algorithme AC-TABOU, qui combine une technique d'arc-consistance à une recherche locale tabou pour explorer l'espace de recherche combinatoire, produit de très bons résultats sur les instances du *FAPP*.

Il y a au moins deux raisons majeures d'effectuer un pré-traitement par filtrage arc-consistant :

- la régularité du comportement de l'algorithme hybride;
- la preuve d'optimalité de k–faisabilité pour la plupart des instances du *FAPP*.

Plusieurs améliorations de l'approche hybride peuvent être envisagées, notamment dans trois directions :

- rendre l'arc-consistance plus rapide par une étude approfondie de AC-3, AC-6, AC-2000,..., algorithmes décris dans [2, 3];
- rendre le filtrage plus efficace (*i.e.* diminuer davantage la taille du domaine de fréquences) en cherchant de bonnes propriétés dans le réseau de contraintes du *FAPP*;
- prendre en compte les deux autres composantes de la fonction objectif f : $|V^{k-1}|$ et $\sum_{j<k-1} |V^j|$.

En conclusion, ce travail nous donne un nouvel exemple de l'efficacité de combiner des méthodes exactes et heuristiques, comme cela a été déjà illustré dans [11, 12], pour faire face à la difficulté des problèmes d'optimisation combinatoire de grande taille.

Références

[1] K.I. Aardal, C.A.J. Hurkens, J.K. Lenstra, and S.R. Tiourine. Algorithms for Frequency Assignment Problems. *CWI, Quartely*, 9:1–9, 1996.

[2] C. Bessière. Arc–consistency and Arc-consistency again. *Artificial Intelligence*, 65:179–190, 1994.

[3] C. Bessière and J.C. Régin. Refining the Basic Consistency Propagation Algorithm. In *the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 309–315, Seattle-Washington,USA, 2001.

[4] V.D. Cung and T. Defaix. Quelques problèmes d'allocation de fréquences et le Challenge ROADEF'2001. *Actes des 7ièmes Journées Nationales sur la résolution pratique de Problèmes NP-Complets (JNPC'2001)*, pages 9–19, 2001.

[5] R. Dorne. *Etude des méthodes heuristiques pour la coloration, la T-coloration et l'affectation de fréquences.* PhD thesis, Université Montpellier II Sciences et Techniques du Languedoc, mai 1998.

[6] C. Fleurent and J.A. Ferland. Genetic and Hybrid Algorithms for Graph Coloring. *Annals of Operations Research*, 63:437–461, 1996.

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

[8] M. Jiang. *Méthodes Approchées pour le Problème de Coloriage Généralisé Application au Problème d'Allocation de Fréquences Multiservices dans l'Aviation Civile.* PhD thesis, Université Versailles–Saint–Quentin–en–Yvelines, novembre 1996.

[9] A. K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[10] R. Mohr and T.C. Henderson. Arc and Path Consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[11] M. Vasquez and J.K. Hao. A Hybrid Approach for the 0–1 Multidimensionnal Knapsack Problem. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence, (IJCAI-01)*, volume 1, pages 328–333, august 2001.

[12] M. Vasquez and J.K. Hao. A "Logic-Constrained" Knapsack Formulation and a Tabu Algorithm for the Daily Photograph Scheduling of an Earth Observation Satellite. *Computational Optimization and Applications*, 20:137–157, 2001.