Projet de Multi Agents System

The CELAR Radio Link Frequency Assignment Problems

Etudiants : BRAHIMI Abdel Madjid - CITAK Bünyamin
Enseignants : PICARD Gauthier
Cursus : Master Données et Systèmes Connectés

Table des matières

Résumé

Comme les problèmes techniques deviennent de plus en plus difficiles et sophistiqués à résoudre avec le temps, les chercheurs s'efforcent d'explorer de nouvelles techniques pour résoudre ces problèmes et les exporter dans des applications réelles. Notre champ d'action porte sur les problèmes à fortes contraintes et l'utilisation de systèmes multi-agents pour les résoudre. Notre considération des multi-agents est due au fait qu'un très grand nombre de variables contraintes sont impliquées dans ces problèmes. Dans notre projet expérimental, nous utilisons FRODO [1], un framework open-source pour l'optimisation des contraintes distribuées.

Le but de notre projet[2] est de comprendre le fonctionnement interne des algorithmes de résolution de contraintes distribuées et de rapporter les meilleures solutions possibles que nous obtenons pour le CELAR [3], pour l'assignation des fréquences des liaisons radio.

1 Introduction

Dans ce projet nous allons utiliser des données fournit par le Centre d'Électronique de l'Armement (CELAR), qui sont le fruit d'un projet européen EUCLID CALMA (Combinatorial Algorithms for Military Applications). Où l'on trouve un ensemble de problèmes de référence pour l'assignation des fréquences de liaison radio (RLFAP) construits à partir d'un réseau réel, avec des données simplifiées. Les contraintes sont toutes binaires, non linéaires et les variables ont des domaines finis. Il s'agit de problèmes de taille réelle, les plus grands instances ayant environ mille variables et plus de cinq mille contraintes. Il y a aussi quelques idées sur l'aspect purement pratique de ces problèmes et plus particulièrement sur l'origine des critères optimisés.

2 Description du problème

Le problème consiste à assigner des fréquences à un ensemble de liaisons radio définies entre des paires de sites afin d'éviter les interférences. Chaque liaison radio est représentée par une variable dont le domaine est l'ensemble des fréquences disponibles pour cette liaison. Les contraintes essentielles concernent deux variables F1 et F2 :

Les deux variables représentent deux liaisons radio qui sont "proches" l'une de l'autre. La constante k_{12} dépend de la position des deux liaisons ainsi que de l'environnement physique. Elle est obtenue à l'aide d'un modèle mathématique de propagation des ondes électromagnétiques qui est encore très "grossier". C'est pour cela que les valeurs peuvent être fausses. Cela ne pose pas de problème, car k_{12} est sur-estimé pour éviter justement le brouillage.

Pour chaque pair de sites, ici, nous allons les nommer A et B, il faut assigner deux valeurs, l'une de A vers B et l'inverse.

$$A \Rightarrow B \text{ et } B \Rightarrow A$$

Ce qui implique la contrainte suivante : la distance en fréquence entre le site $A \Rightarrow B$ et $B \Rightarrow A$ doit être exactement égale à 238.

La possibilité d'exprimer des contraintes telles que $|F1 - F2| > k_{12}$ réussit à exprimer le problème de coloration du graphe et il est donc clair que le RLFAP est NP-Hard.

3 Critères d'optimisation

Pour évaluer les assignations de fréquence obtenues, nous avons besoin de normes sur lesquelles nous basons notre jugement de la solution obtenue. Dans le cadre de notre projet, nous nous concentrons sur deux critères principaux. Premièrement, la minimisation des valeurs de fréquence. En d'autres termes, les fréquences assignées aux liaisons de données doivent être réduites au minimum pour des raisons de consommation d'énergie technique. Le deuxième critère est la minimisation du nombre de fréquences utilisées, car il y aura des liaisons de données éloignées les unes des autres afin qu'elles puissent utiliser les mêmes fréquences. Cela facilite l'installation de futures liaisons de données avec les mêmes domaines de fréquences.

4 Présentation des fichiers

4.1 var.txt

Ce fichier possède toutes les variables que nous allons utiliser pour créer les agents, car nous avons choisi d'assigner une variable à un agent. Dans ce fichier qui est de la manière suivante :
1 0
2 7
...
32 4

Où la première colonne est la variable et la seconde est le domaine auquel il appartient. On s'aperçoit qu'il y a 8 domaines numérotés de 0 à 7. Dans tous les fichiers dom.txt on retrouve les mêmes valeurs.

4.2 dom.txt

Ce fichier nous donne l'ensemble des valeurs pour chaque domaine. Ce qui veut dire par exemple que si l'on veut que $|var1 - var2| > 34$ et $|var4 - var2| > 76$ on va prendre les valeurs qui composent les domaines des variables pour y respecter les deux contraintes.

Dans ce fichier qui est de la manière suivante :
0 6 23 34 45 56 57 68
...
...
7 2 11 789
Où la première colonne est le numéro du domaine, la seconde est le cardinal du domaine et la suite sont les valeurs qui composent ce domaine.

4.3 ctr.txt

Ce fichier est l'un des plus important car c'est à grâce à lui que l'on sait les contraintes qui sont appliquées entre le lien de deux sites. C'est ici qu'on apprend la valeur k_{12} et où l'on va apprendre la valeur du coût C que l'on va expliquer dans la section suivante. Dans ce fichier, qui n'est pas toujours complet, il y a cinq informations que l'on retrouve toujours, les voici :
- 1 2 D = 238
- 1 171 L > 8

Que l'on peut traduire par $|1 - 2| = 238$ et $|1 - 171| > 8$

Où la première colonne est la première variable qui est en lien avec la seconde. La troisième colonne ne nous ait pas utile pour la suite. La quatrième colonne est l'opérateur à utiliser pour vérifier la contrainte. Et la cinquième colonne est le résultat à obtenir pour cette contrainte. Comme nous l'avons vu dans une section précédente, pour l'opérateur égalité, il faut que la contrainte vaut 238.

Dans certains fichiers, nous avons une sixième colonne qui représente le coût que l'on attribue à cette contrainte. Ces coûts ont les retrouve dans le fichier 'cst.txt'

4.4 cst.txt

Dans ce fichier, nous retrouvons les informations nécessaires pour créer notre fichier xml. Dans certains cas, il n'y a aucune information sur les coûts possibles, mais des fois nous avons des informations que nous devons utiliser. Si ce fichier possède des coûts, on les utilisera dans nos fonctions pour le calcul des coûts des contraintes.

5 Explication de la fonction de coût

Pour résoudre ce problème (RLFAP), nous avions besoin d'une fonction de coût. Nous avons créé notre propre fonction de coût. Le but de cette démarche est d'adapter le coût résultant de chaque contrainte afin de minimiser les valeurs des fréquences et de minimiser également le nombre de fréquences utilisées. La raison de cette décision est qu'elle rend la partie de développement plus facile à déboguer, elle donne une analyse plus claire de la performance de la fonction objectif et enfin elle conduit l'investigation pour savoir si nous pouvons atteindre une solution assez coûteuse si le problème est irréalisable ou non. Notre fonction objective de coût vise la minimisation du nombre de fréquences comme formule :

$$if(|F1 - F2| \geq K_{12}), 0, C. \tag{1}$$

Ici, C correspond à la sixième colonne du fichier 'ctr.txt', si la valeur existe, on récupère la valeur associé à ce nombre dans le fichier 'cst.txt', sinon on lui attribue 1.

Les coûts choisis sont de la sorte :

$$0 => a1 = 1000 \tag{2}$$

$$1 => a2 = 100 \tag{3}$$

$$2 => a3 = 10 \tag{4}$$

$$3 => a4 = 1 \tag{5}$$

$$4 => b1 = 1 \tag{6}$$

À partir de cela, nous choisirons en fonction du fichier 'cst.txt' les valeurs que nous voulons attribuer aux valeurs (0,1,2,3,4).

6 Les méthode mise en place pour créer et exécuter nos fichiers XML

Pour parvenir aux résultats escomptés, nous avons voulu faire le projet tout en python[4]. Car il est plus aisé de faire de manipuler des données scientifiques en python qu'en Java. De plus, cela nous a permis d'apprendre un nouveau langage de programmation. Pour parser les fichiers données, nous avons utilisées les dictionnaires, car nous avons pensé que pour une personne extérieure, comprendre le code sera plus aisé. De plus, une fois avoir obtenu

un dictionnaire par fichier. Nous avons pu y créer de façon rapide et efficace les fichiers xml associer à chaque scen. Nous avons utilisé une bibliothèque Python nommée LXML[5] pour construire les fichiers XML représentant les problèmes d'entrée pour FRODO.

Tous les fichiers de coûts ne sont pas identiques, comme dit précédemment, il y a des fichiers qui ne possèdent pas de "coûts" a proprement dit. Donc on les a mises en durs égale à 1 dans le code, donc c'est facile de les changer selon ce qu'on veut faire. Nous allons expliquer la construction de nos fichiers XML dans le format présenté dans les figures qui suit : les variables (qui sont aussi les agents) et les domaines [6], les prédicats [7] et enfin les contraintes [8]

Une fois le fichier xml obtenue il passe dans une fonction qui appelle frodo en ligne de commande. Dans cette commande, nous avons fixé un timeout de 36 000 000 000 ms ce qui correspond à 10 000 h soit plus d'année. Malgré cela, nous n'arrivons pas à faire tourner les algorithmes suivant :

- DPOP sur toutes les scen
- ADOPT idem DPOP
- AFB idem
- MGM réussi à faire tourner sur scen02 et scen06
- DSA idem MGM
- MaxSum réussi à faire tourner sur scen02 et scen03

De plus, même ces résultats n'ont pas été obtenus via la machine suivante : MacBook Air ram : 8gb et processeur : 1,8 GHz Intel Core i5 double cœur. Ils ont été obtenus via Google Colab, qui disposait d'un GPU (aucune information là-dessus) et 25gb de ram.

C'est pour cela que nous avons pris l'initiative pour avoir plus de résultats de mettre en place des fonctions pour créer des fichiers 'var.txt', 'dom.txt', 'ctr.txt', 'cst.txt'. Ce dernier, pour que cela soit plus simple à déboguer, c'est un fichier sans $a_1, a_2, ...$

7 Résultats

Grâce-au générateur de problème aléatoire, nous avons créé 3 problèmes.

- le premier avec 4 variables, 7 domaines et 12 contraintes.
- le second avec 10 variables, 7 domaines et 12 contraintes.
- le troisième avec 20 variables, 7 domaines et 22 contraintes.

Dans les fichiers CELAR, nous avons pu exécuter scen02, scen03 et scen06 seulement.

Voici le tableau résumant nos résultats :

Algorithm	file	number of variables	time	cost	Number of messages sent (by type)	Amount of information sent
DPOP	random	4	74	3	32	9516
ADOPT	random	4	136	3	1266	54974
AFB	random	4	246	3	1974	72192
DSA	random	4	235	3	1990	41940
MGM	random	4	277	3	4000	125120
MaxSum	random	4	85	6	45	9827
MGM	random	10	723	26	26400	792986
DSA	random	10	664	25	13134	277264
AFB	random	10	4333	25	28049	2736668
DPOP	random	10	2183	25	168	348633
ADOPT	random	10	out	/	/	/
MaxSum	random	10	657	30	3037	359276
DPOP	random	20	out	/	/	/
AFB	random	20	out	/	/	/
DSA	random	20	5323	113	69650	1471727
MGM	random	20	4163	109	140000	4381154
DSA	scen02	200	37731	8	491530	10399038
MGM	scen02	200	35158	21	988000	30925864
MaxSum	scen02	200	10108	1011	3705	2587586
DSA	scen06	200	39746	8505	526156	11116883
MGM	scen06	200	38919	7474	1057600	33102422
MaxSum	scen03	400	14391	2246	8280	5769694

TABLE 1 – Résultats obtenues pour les algorithmes en fonction du nombre de variables

On remarque que pour des petits fichiers DPOP est plus rapide que MGM et DSA. Qu'il envoie beaucoup moins de fichiers que MGM et DSA. Cependant, on se rend compte que plus le nombre de variables augmentent et plus il perd de la place face à MGM et DSA.

À partir de 10 variables, on s'aperçoit que ADOPT, met beaucoup trop de temps pour résoudre le problème. Donc il n'est pas à choisir pour un petit ou grand nombre de problèmes. Sinon cela voudrait dire que le problème n'est pas fait pour du ADOPT. Donc, pour des petits problèmes entre ADOPT et DPOP, il faut choisir DPOP, car il est complet et plus rapide.

À partir de 20 variables, on s'aperçoit que les algorithmes complets ne peuvent marcher. Donc, à partir de ce seuil, on essaie plus que MGM, DSA et MaxSum.

Nous nous sommes aperçus que seul deux scen on pu être testé, les autres donnant des time out (on ne compte pas scen03, car c'est MaxSum seulement donc pas de comparaison.). Parmi ces deux scen, nous avons la 02 et 06. Nous nous apercevons que le temps est à peu près identique entre scen02 et scen06. Mais qu'on ne trouve pas le même coût pour les deux algorithmes. Donc, on peut dire que cela nous à une autre question : lequel est-il meilleur ?

La réponse, cela dépend des nombres de contraintes et les valeurs assigné aux variables au début de l'algorithme.

7.1 Graphique à partir des fichiers créés

Ici, nous allons parler de ce qu'on rajoute dans le projet, hormis le faite que le tableau vu précédemment soit aussi généré par notre programme. Nous pouvons afficher les graphiques correspondant par catégorie pour chaque algorithme testé. Nous allons ici montrer la comparaison que nous obtenons après la création d'un fichier avec 4 variables.

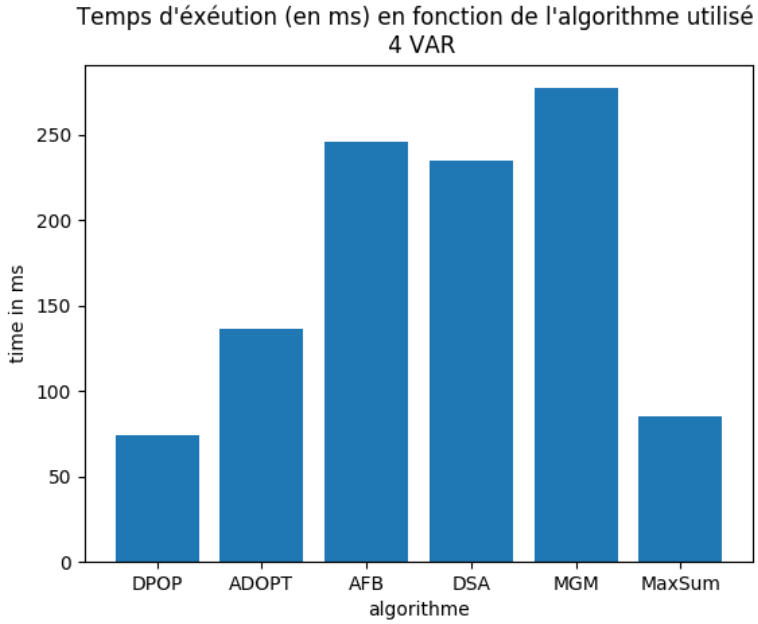

FIGURE 2 – Temps en fonction de l'algorithme

Dans ce graphique on s'aperçoit des différences entre les algorithmes complets et non-complets. DPOP est presque que trois plus rapides que DSA et MGM.

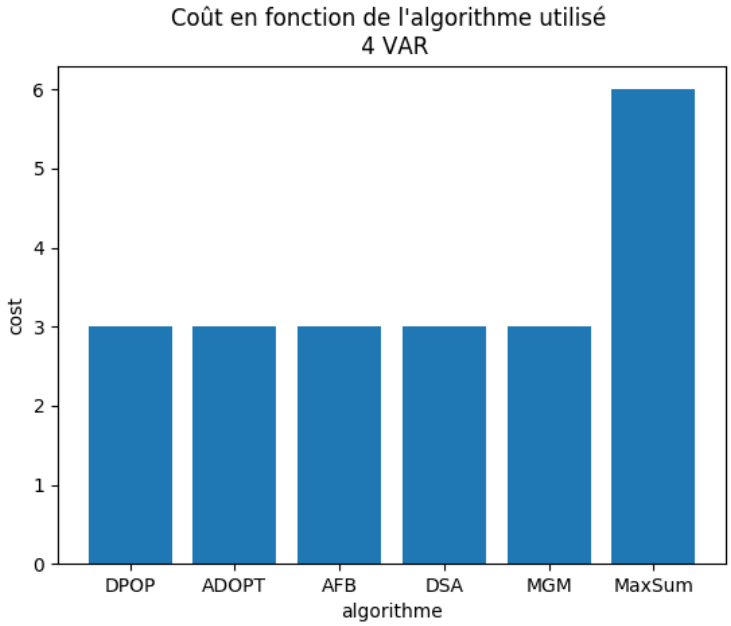

FIGURE 3 – Coût en fonction de l'algorithme

Ici, on s'aperçoit qu'ils ont tous le même coût sauf MaxSum. D'ailleurs, cet algorithme renvoie toujours des coûts "faux". Mais quand on veut aller vite et que le problème du coût n'est pas 'si important' on se permettre de l'utiliser. Sinon, nous le déconseillons.

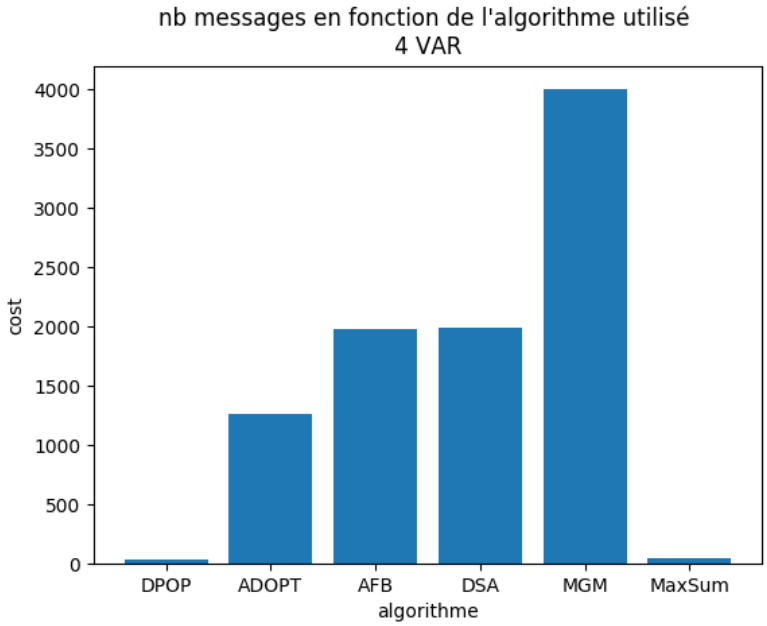

FIGURE 4 – Nombre de messages envoyés en fonction de l'algorithme

La différence est beaucoup plus marquante grâce-à ce graphique. On voit que DPOP peut trouver le coût optimal en très peu de messages envoyés. Alors que pour le même problème MGM envoie près de 4 000 messages.

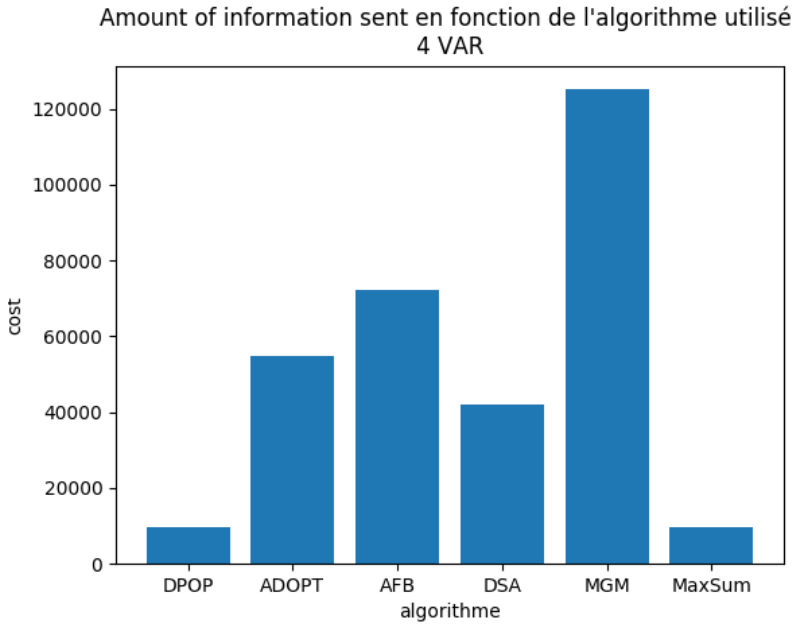

FIGURE 5 – Nombre d'informations envoyés en fonction de l'algorithme

Ici, on peut dire que par rapport au graphique d'avant que DSA envoie moins d'information en amont qu'ADOPT et AFB alors qu'il envoie plus de messages.

8 Conclusion

Pour conclure, nous pouvons dire que la résolution du problème de Radio Link Frequency Assignment est très difficile et demande des ressources dont nous ne disposons pas (Rappel : problème NP-Hard). Mais ce qui est assez surprenant après autant de test, c'est de voir à quel point DPOP est efficace, mais qu'il est contraint par le nombre de variables. De plus, on a vu que même si MGM et DSA arrivait à donner des résultats pour des problèmes à 200 variables, quand il s'agit de petit problème, ils ne sont pas si efficaces, même si à priori nous pouvons se dire le contraire.

Bibliographie

[1] Thomas Léauté, Brammert Ottens, and Radoslaw Szymanek. FRODO 2.0 : An open-source framework for distributed constraint optimization. In *Proceedings of the IJCAI'09 Distributed Constraint Reasoning Workshop (DCR'09)*, pages 160–164, Pasadena, California, USA, July 13 2009. http://www7.inra.fr/mia/T/schiex/Doc/CELAR.shtml.

[2] Gauthier Picard. The celar radio link frequency assignment problems. November 2019. https://www.emse.fr/~picard/cours/mas/MAS-project-2019.pdf.

[3] Thomas Schiex. The celar radio link frequency assignment problems. https://frodo-ai.tech.

[4] Learned by example. https://www.learnbyexample.org/python/.

[5] Stephan Richter. lxml - xml and html with python. November 25 2019. https://lxml.de.

```
<agents nbAgents="10">
  <agent name="agent1"/>
  <agent name="agent2"/>
  <agent name="agent3"/>
  <agent name="agent4"/>
  <agent name="agent5"/>
  <agent name="agent6"/>
  <agent name="agent7"/>
  <agent name="agent8"/>
  <agent name="agent9"/>
  <agent name="agent10"/>
</agents>
<domains nbDomains="7">
  <domain name="dom1" nbValues="7">560 574 215 627 24 589 679</domain>
  <domain name="dom2" nbValues="1">86</domain>
  <domain name="dom3" nbValues="8">791 729 42 95 350 402 746 76</domain>
  <domain name="dom4" nbValues="7">468 719 477 523 424 783 275</domain>
  <domain name="dom5" nbValues="6">320 202 748 145 49 523</domain>
  <domain name="dom6" nbValues="11">227 487 660 697 235 184 316 435 525 367 652</domain>
  <domain name="dom7" nbValues="3">71 107 677</domain>
</domains>
<variables nbVariables="10">
  <variable name="var1" domain="dom5" agent="agent1"/>
  <variable name="var2" domain="dom7" agent="agent2"/>
  <variable name="var3" domain="dom7" agent="agent3"/>
  <variable name="var4" domain="dom5" agent="agent4"/>
  <variable name="var5" domain="dom3" agent="agent5"/>
  <variable name="var6" domain="dom3" agent="agent6"/>
  <variable name="var7" domain="dom1" agent="agent7"/>
  <variable name="var8" domain="dom3" agent="agent8"/>
  <variable name="var9" domain="dom2" agent="agent9"/>
  <variable name="var10" domain="dom6" agent="agent10"/>
</variables>
```

FIGURE 6 – Variables, agents et domaines

```
<predicates nbPredicates="2">
  <predicate name="gt">
    <parameters>int variable1 int variable2 int K int C</parameters>
    <expression>
      <functional>if(gt(abs(sub(variable1,variable2)),K),0,C)</functional>
    </expression>
  </predicate>
  <predicate name="eq">
    <parameters>int variable1 int variable2 int K int C</parameters>
    <expression>
      <functional>if(eq(abs(sub(variable1,variable2)),K),0,C)</functional>
    </expression>
  </predicate>
</predicates>
```

FIGURE 7 – Prédicat

```
</constraint>
<constraint name="1_gt_5_with_K=372" arity="2" scope="var1 var5" reference="gt">
  <parameters>var1 var5 372 1</parameters>
</constraint>
<constraint name="1_eq_6_with_K=238" arity="2" scope="var1 var6" reference="eq">
  <parameters>var1 var6 238 1</parameters>
</constraint>
```

FIGURE 8 – Contraintes