Mini-projet OPL - MPRO 2022-2023

Camille Richer, Héloïse Gachet

Modélisation et Gestion des symétries

Proposer différentes modélisations pour ces problèmes ; expliquer les avantages et inconvénients de chacune. Quels sont les points clés à considérer pour optimiser les temps de recherche de solution sur les problèmes 2 et 3 ?

Problème 1 : Allocation de fréquences

On cherche à affecter des fréquences (notés par leur canal 1 à 10) à 7 émetteurs, en s'assurant que les émetteurs pairs ont des fréquences impaires et les émetteurs impairs des fréquences paires (choix fait en constatant l'ambiguïté dans le sujet) et en s'assurant que si deux émetteurs T_1 et T_2 sont voisins, alors leurs fréquences vérifient $|f_{T_1} - f_{T_2}| \geq \text{offset}_{T_1,T_2}$ où offset_{T_1,T_2} est donné par une table pour toute paire de voisins. Deux modélisations sont possibles. On peut prendre des variables de décision binaires

$$x_{fe} = \begin{cases} 1 & \text{si la fréquence } f \text{ est affectée à l'émetteur } e \\ 0 & \text{sinon.} \end{cases}$$

Dans ce cas le problème s'écrit :

$$\begin{cases} \min & \max\limits_{e}(\sum\limits_{f=1}^{10} fx_{fe}) & \\ s.c. & \sum\limits_{f=1}^{10} x_{fe} = 1 & \forall e \in \{1,\dots,7\} \\ & x_{fe} = 0 & \forall (f,e), f \text{ pair et } e \text{ pair}, \forall (f,e), f \text{ impair et } e \text{ impair} \\ & |\sum\limits_{f=1}^{10} fx_{fe_1} - \sum\limits_{f=1}^{10} fx_{fe_2}| \geq \text{offset}_{e_1,e_2} & \forall (e_1,e_2) \text{ voisins} \end{cases}$$

Mais cette formulation sert plutôt à faire de la programmation linéaire. Une formulation plus adaptée pour la PPC serait de prendre une variable x_e par émetteur, de domaine $\{1,\dots,10\}$, et qui vaut la fréquence assignée à l'émetteur e. Le problème s'écrit très simplement :

$$\begin{cases} \min & \max\limits_{e}(x_e) & \\ s.c. & x_e \equiv 1(2) & \forall e \equiv 0(2) \text{ (filtrage des domaines)} \\ & x_e \equiv 0(2) & \forall e \equiv 1(2) \text{ (filtrage des domaines)} \\ & |x_{e_1} - x_{e_2}| \geq \text{offset}_{e_1,e_2} & \forall (e_1,e_2) \text{ voisins} \end{cases}$$

Problème 2 : Cavaliers

Le problème consiste à positionner un nombre minimum de cavaliers sur un échiquier de taille 8 de telle sorte que chaque case soit contrôlée (occupée ou menacée) par au moins un cavalier. Comme on ne sait pas à l'avance combien de cavaliers on va placer, on ne peut pas rechercher directement leurs coordonéees, on choisit donc d'avoir une variable binaire par case de l'échiquier :

$$x_{ij} = \begin{cases} 1 & \text{si la case } (i,j) \text{ comporte un cavalier} \\ 0 & \text{sinon.} \end{cases}$$

La condition de couverture pour une case s'écrit :

$$\forall (i,j), x_{i,j} + x_{i-2,j+1} + x_{i-1,j+2} + x_{i+1,j+2} + x_{i+2,j+1} + x_{i+2,j-1} + x_{i+1,j-2} + x_{i-1,j-2} + x_{i-2,j-1} \geq 1$$

Mais pour qu'elle soit valide pour toutes les cases de l'échiquier, on considère qu'on ajoute deux rangées de chaque côté (plateau de côté 12). On ne peut pas placer de cavalier sur les cases artificielles, mais elles n'ont pas besoin d'être couvertes par un cavalier. Le problème s'écrit alors :

$$\begin{cases} \min & \sum_{i,j}(x_{ij}) & \\ s.c. & x_{i,j} + x_{i-2,j+1} + x_{i-1,j+2} + x_{i+1,j+2} + x_{i+2,j+1} & \\ & +x_{i+2,j-1} + x_{i+1,j-2} + x_{i-1,j-2} + x_{i-2,j-1} \geq 1 & \forall i,j \in \{3,\ldots,10\} \\ & x_{ij} = 0 & \forall i \text{ ou } j \in \{1,2,11,12\} \end{cases}$$

Comme il s'agit d'un problème d'optimisation, la programmation par contrainte va le résoudre itérativement. Il faut donc s'assurer que chaque problème individuel n'est pas trop long à résoudre. Pour ça, on peut remarquer qu'une solution optimale peut être symétrique, et se borner à explorer les solutions symétriques. On ajoute donc les contraintes :

$$x_{i,j} = x_{12-i+1,12-j+1} \forall i,j \in \{1,\ldots,12\}$$

Il y a des redondances mais ça n'est pas gênant.

Implémentation

Implémenter les modèles choisis avec IBM ILOG CPLEX Optimisation studio ; tester. Bien séparer les données du modèle (.dat pour les données ; .mod pour les modèles) ; utiliser les facilités de modélisation offertes par le langage pour écrire un modèle le plus concis et plus générique possible.

Problème 1 : Allocation de fréquences

Les différentes questions ont été divisées en fichiers et configurations d'exécution.

Configuration d'exécution 1 : Une recherche de solution réalisable donne : [6 9 6 3 2 9 2]. Une recherche en utilisant `minimize` donne : [6 1 6 3 4 1 4] soit une fréquence maximale utilisée de 6.

Configuration d'exécution 2 : La recherche des 10 premières solutions donne :

```
[8 3 8 1 6 3 6]
[8 3 8 1 6 5 6]
[8 3 8 1 6 7 6]
[8 3 8 1 6 9 6]
[8 3 8 1 6 3 10]
[8 3 8 1 6 5 10]
[8 3 8 1 6 7 10]
[8 3 8 1 6 9 10]
[10 3 8 1 6 3 6]
[10 3 8 1 6 5 6]
```

Configuration d'exécution 3 : La recherche d'une solution optimale par approche branch and bound donne : [6 9 6 3 2 9 2] → [8 1 6 3 4 5 8] → [2 7 2 5 4 1 4] → [6 1 6 3 4 5 4]. On obtient une solution optimale différente de la première mais avec la même fréquence maximale allouée.

Problème 2 : Cavaliers

La recherche d'une solution optimale avec `minimize` donne (les 0 indiquent une case avec un cavalier et les X une case sans) :

```
// solution with objective 12
X X X X X X X X
X X 0 X X X X X
X X 0 0 X 0 0 X
X X X X X 0 X X
X X 0 X X X X X
X 0 0 X 0 0 X X
X X X X X 0 X X
X X X X X X X X
```

Recherche de solutions et Propagation des contraintes

Essayer différents paramétrages de recherche de solution (fichiers params), et écrire des sélecteurs spécifiques avec les options de searchPhase vues dans la présentation de CPLEX Studio. Créer des configurations d'exécution pour les différents paramétrages. Notez les indicateurs de recherche (onglets Journal du moteur et statistiques). Tester, présenter les résultats. Discuter.

Essayer différents niveaux de propagation des contraintes (fichiers param). Notez les indicateurs de recherche (onglets Journal du moteur et statistiques); Discuter.

Le paramétrage par défaut est :

Presolve	on
Default inference level	Basic
Search type	Auto
Number of workers	-1

Problème 1 : Allocation de fréquences

Les tests ont été effectués avec le fichier `.mod` utilisant la méthode `minimize` et les différentes configurations d'exécution portent le même nom que le fichier `.ops` qu'elles contiennent. Le modèle comporte 7 variables et 21 contraintes. Par défaut, la résolution utilise 8 workers. Le paramètre qu'on fait varier est indiqué entre parenthèses dans la colonne de gauche.

parameters	time spent in solve	number of branches	number of fails
default	0.01s	8 675	4 754
param1 (solve off)	0,01s	8 675	4 754
param2 (inference level extended)	0,04s	8 675	4 754
param3 (search type depth first)	0,00s	148	60
param4 (search type restart)	0,01s	1 175	356
param5 (search type iterative diving)	0,04s	18 688	10 257
param 6 (10 workers)	0,01s	10 855	5 951

On constate que le paramètre qui a le plus d'influence est la stratégie de recherche (profondeur d'abord ici). Comme on sait qu'on recherche une solution qui minimise la fréquence maximum allouée, on peut définir une searchPhase pour sélectionner la plus petite valeur en premier lors d'une recherche en profondeur d'abord. Pour le choix de variable, on peut prendre celle de plus petit domaine, et comme on a constaté que plus de workers ne permettait pas d'améliorer le résultat en temps, on peut en prendre un seul pour minimiser les branchements.

Configuration d'exécution SearchPhase : on obtient la solution optimale [6 1 6 3 4 1 4] avec les indicateurs :

parameters	time spent in solve	number of branches	number of fails
search phase	0,01s	20	8

Problème 2 : Cavaliers

Pour le problème des cavaliers, on peut essayer de constater l'influence de l'ajout des contraintes de symétrie sur le temps de résolution : on utilise dans les deux cas `minimize`.

	number of constraints	time spent in solve	number of branches	number of fails
avec symétries	310	0,21s	167 064	82 317
sans symétries	165	1:20s (stop)	68 957 408	33 794 591

Sans mettre de contraintes de symétrie, le solveur tourne beaucoup trop longtemps car même s'il a trouvé une solution de valeur 12 (valeur optimale) en très peu de temps (environ 0,625s), il continue d'explorer. Les contraintes de symétrie sont donc indispensables.

On peut ensuite tester l'influence des autres paramètres comme pour le problème 1 :

parameters	time spent in solve	number of branches	number of fails
default	0,19s	167 064	82 317
param1 (presolve off)	0,27s	205 181	100 869
param2 (inference level extended)	0,19s	167 064	82 317
param3 (search type depth first)	0,15s	100 583	50 233
param4 (search type restart)	0,21s	180 381	87 251
param5 (search type iterative diving)	0,18s	150 011	73 617
param 6 (10 workers)	0,21s	209 487	102 977

Ici aussi, la meilleure stratégie de recherche est en profondeur d'abord et les workers multiples ne servent à rien. Lors d'une exécution avec le presolve, le moteur renvoie `! Presolve : 58 extractables eliminated` indiquant qu'il a trouvé une formulation plus compacte du problème, et cela se ressent sur le temps d'exécution. On peut donc définir une SearchPhase en profondeur d'abord, avec un seul worker, et comme on cherche à minimiser le nombre de cavaliers, on peut essayer d'assigner en premier la valeur 0.

Configuration d'exécution SearchPhase : on obtient la solution optimale

```
X X X X X X X X
X X X X X 0 X X
X 0 0 X 0 0 X X
X X 0 X X X X X
X X X X X 0 X X
X X 0 0 X 0 0 X
X X 0 X X X X X
X X X X X X X X
```

avec les indicateurs :

parameters	time spent in solve	number of branches	number of fails
search phase	0,12s	11 208	5 603

Cela permet d'améliorer un peu les performances du solveur.

Conclusion

La programmation par contrainte est efficace quand la formulation du problème s'y prête bien, ce qui n'est pas toujours le cas, et la stratégie de recherche a une grande influence sur et elle n'est pas particulièrement adaptée pour trouver une solution optimale car on doit le faire itérativement. En comparaison, pour le problème 2, la résolution en programmation mathématique avec le paramétrage par défaut trouve la même solution que la configuration SearchPhase mais en 0,02s et à la racine.