INSTITUT POLYTECHNIQUE DE PARIS

Le Cnam
Institut Polytechnique de Paris

Utilisation d'OPL pour la PPC

Auteurs :
Justine De Sousa
Vincent Maron

Superviseur :
Eric Nespoulous

le cnam

13 janvier 2022

1 Allocation de fréquences (PPC)

Dans ce problème, il s'agit d'allouer des fréquences à des transmetteurs tout en évitant les interférences de transmetteurs proches et en minimisant la plage de fréquences empruntée. Pour ce faire, les fréquences sont modélisées par des entiers majorés par une valeur, qu'on note D. Pour tout couple de transmetteurs, on modélise l'absence d'interférences par un écart minimum imposé entre leurs deux fréquences, noté offset(i, j), relativement aux transmetteurs i et j. De plus, on impose que les transmetteurs pairs aient des fréquences paires, et les transmetteurs impairs aient des fréquences impaires, les transmetteurs étant indexés par un entier, et en supposant qu'ils sont au nombre de N.

1.1 Modélisation

Le premier modèle est très naïf, il s'agit d'une traduction directe du problème. Il a l'avantage d'être simple et clair.

Dans le second modèle, on restreint le domaine des variables d'indice pair et impair plutôt que de mettre des contraintes sur la parité des fréquences. Cela a l'avantage de réduire le domaine des variables de moitié et donc de diminuer le nombre de noeuds à explorer. Il y a également moins de contraintes et donc une exécution plus rapide en chaque noeud de l'arborescence. En terme d'implémentation, c'est par contre moins clair, il y a donc un post-processing à effectuer pour l'affichage de la solution.

Premier modèle

- Variables : $x_1, ..., x_N \in \{1, ..., D\}$
- Contraintes :
  - $\star$ $\forall\, i \in \{1, ..., N\} \quad x_i\,[2] = i\,[2]$
  - $\star$ $\forall\, i, j \in \{1, ..., N\} \quad |x_i - x_j| \geqslant \text{offset}(i, j)$
- Objectif : $\min \max_{i \in \{1,...,N\}} x_i$

Modèle alternatif

- Variables : $x_1, ..., x_N$
  - $\star$ $\forall\, i$ impair $x_i \in 2\,\{1, ..., \lceil D/2 \rceil\} - 1$
  - $\star$ $\forall\, i$ pair $x_i \in 2\,\{1, ..., \lfloor D/2 \rfloor\}$
- Contraintes :
  - $\star$ $\forall\, i, j \in \{1, ..., N\} \quad |x_i - x_j| \geqslant \text{offset}(i, j)$

Comparons ces deux modèles dans une configuration d'exécution par défaut (Figure 1 et Figure 2). Lors

```
! -----------------------------------------------------------------
! Minimization problem - 7 variables, 58 constraints
! Initial process time : 0,02s (0,02s extraction + 0,00s propagation)
!  . Log search space  : 22,2 (before), 22,2 (after)
!  . Memory usage      : 299,1 kB (before), 299,1 kB (after)
! Using parallel search with 4 workers.
! -----------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
                       0          7                -
+ New bound is 2
                       0          7    1   F       -
+ New bound is 5
                       2          7    1   F     2 != x(2)
+ New bound is 6
*              9        8  0,04s        1      (gap is 33,33%)
*              8       42  0,04s        1      (gap is 25,00%)
*              7      149  0,04s        1      (gap is 14,29%)
*              6      153  0,04s        1      (gap is 0,00%)
! -----------------------------------------------------------------
! Search completed, 4 solutions found.
! Best objective         : 6 (optimal - effective tol. is 0)
! Best bound             : 6
! -----------------------------------------------------------------
! Number of branches     : 2 499
! Number of fails        : 1 272
! Total memory usage     : 2,6 MB (2,2 MB CP Optimizer + 0,4 MB Concert)
! Time spent in solve    : 0,05s (0,03s engine + 0,02s extraction)
! Search speed (br. / s) : 73 500,0
! -----------------------------------------------------------------
```

FIGURE 1 – Modèle naïf

```
! -----------------------------------------------------------------
! Minimization problem - 7 variables, 50 constraints
! Initial process time : 0,01s (0,01s extraction + 0,00s propagation)
!  . Log search space  : 16,3 (before), 16,3 (after)
!  . Memory usage      : 299,5 kB (before), 299,5 kB (after)
! Using parallel search with 4 workers.
! -----------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
                       0          7                -
+ New bound is 2
                       0          7    1   F       -
+ New bound is 5
                       2          7    1   F     1 != even(1)
+ New bound is 6
*              9        8  0,03s        1      (gap is 33,33%)
*              7       47  0,03s        1      (gap is 14,29%)
*              6      143  0,03s        1      (gap is 0,00%)
! -----------------------------------------------------------------
! Search completed, 3 solutions found.
! Best objective         : 6 (optimal - effective tol. is 0)
! Best bound             : 6
! -----------------------------------------------------------------
! Number of branches     : 2 368
! Number of fails        : 1 216
! Total memory usage     : 2,7 MB (2,2 MB CP Optimizer + 0,4 MB Concert)
! Time spent in solve    : 0,04s (0,02s engine + 0,01s extraction)
! Search speed (br. / s) : 84 571,4
! -----------------------------------------------------------------
```

FIGURE 2 – Modèle alternatif

de la résolution, les deux modèles trouvent la même solution : 1 6 1 4 3 2 3. Il y a autant de variables dans chacun des modèles et plus de contraintes dans le modèle naïf. Le modèle alternatif visite moins de noeuds que le modèle naïf et il visite 3 solutions (contre 4 pour le modèle naïf) avant de trouver la solution optimale. La résolution est donc plus rapide avec le modèle alternatif. Dans la suite, on conserve le modèle alternatif.

Description des fichiers

- `freq_alloc_sol_real.mod` : 1er modèle naïf permettant de trouver une solution réalisable.
- `freq_alloc_alternative.mod` : recherche de solution avec le modèle alternatif.
- `freq_alloc_minimize.mod` : minimisation de la solution avec le mot clef `minimize`.
- `freq_alloc_10_sol.mod` : affichage des 10 premières solutions trouvées par le solveur avec le mot clef `cp.next()`.
- `freq_alloc_boucle_min.mod` : minimisation de la solution avec une procédure de type Branch&Bound.

1.2 Minimisation "à la main"

On écrit une boucle `while` qui, à chaque itération, tente de résoudre un nouveau problème de décision associé à une fréquence maximum qui décrémente. Lorsque le modèle n'a plus de solution, cela signifie que la dernière solution trouvée était une solution optimale du problème d'optimisation. Lors de l'exécution de ce script, 5 problèmes de décision sont résolus avec les fréquences max qui valent 10, 8, 7 , 6 et 5. Les 4 premiers renvoient une solution, le dernier n'en renvoie pas. Les solutions optimales de ce problème ont donc 6 fréquences attribuées. La solution trouvée avec cette procédure est la même que précédemment.

1.3 Le type de recherche

On reprend la minimisation par défaut d'OPL avec le mot clef `minimize`. On compare différentes options de recherche (Figure 3, Figure 4, Figure 5 et Figure 6). On remarque que toutes les méthodes trouvent au moins

```
! ------------------------------------------------------------------
! Minimization problem - 7 variables, 50 constraints
! TimeLimit            = 0,1
! LogVerbosity         = Verbose
! SearchType           = DepthFirst
! Initial process time : 0,00s (0,00s extraction + 0,00s propagation)
!  . Log search space  : 16,3 (before), 16,3 (after)
!  . Memory usage      : 299,5 kB (before), 299,5 kB (after)
! Using parallel search with 4 workers.
! ------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
                       0          7                -
+ New bound is 2
*              9        6  0,01s        1      (gap is 77,78%)
*              7       10  0,01s        1      (gap is 71,43%)
*              6       17  0,01s        1      (gap is 66,67%)
! ------------------------------------------------------------------
! Search completed, 3 solutions found.
! Best objective         : 6 (optimal - effective tol. is 0)
! Best bound             : 2
! ------------------------------------------------------------------
! Number of branches     : 72
! Number of fails        : 28
! Total memory usage     : 2,4 MB (1,9 MB CP Optimizer + 0,5 MB Concert)
! Time spent in solve    : 0,02s (0,01s engine + 0,00s extraction)
! Search speed (br. / s) : 5 538,5
! ------------------------------------------------------------------
```

Figure 3 – DepthFirst

```
! ------------------------------------------------------------------
! Minimization problem - 7 variables, 50 constraints
! TimeLimit            = 0,1
! LogVerbosity         = Verbose
! SearchType           = MultiPoint
! Initial process time : 0,01s (0,01s extraction + 0,00s propagation)
!  . Log search space  : 16,3 (before), 16,3 (after)
!  . Memory usage      : 299,5 kB (before), 299,5 kB (after)
! Using parallel search with 4 workers.
! ------------------------------------------------------------------
! Time = 0,08s, Average fail depth = 18, Memory usage = 2,5 MB
! Current bound is 2 (gap is 66,67%)
!          Best Branches  Non-fixed    W       Branch decision
              6    4 000          7    3            -
              6    4 000          7    4            -
              6    5 000          7    1            -
! ------------------------------------------------------------------
! Search terminated by limit, 5 solutions found.
! Best objective         : 6 (gap is 66,67%)
! Best bound             : 2
! ------------------------------------------------------------------
! Number of branches     : 22 889
! Number of fails        : 12 783
! Total memory usage     : 2,9 MB (2,4 MB CP Optimizer + 0,5 MB Concert)
! Time spent in solve    : 0,11s (0,10s engine + 0,01s extraction)
! Search speed (br. / s) : 215 934,0
! ------------------------------------------------------------------
```

Figure 4 – multiPoints

une solution de valeur 6. Les méthodes `DepthFirst` et `Restart` sont les plus rapides à renvoyer leur solutions. Ce sont également les méthodes qui parcourent le moins de branches. `IterativeDiving` a un comportement similaire à celui par défaut et enfin la méthode `MultiPoints` est la dernière à renvoyer sa solution. Elle s'arrête uniquement par limite de temps. Elle trouve néanmoins 5 solutions dont au moins une de valeur 6. Cette méthode parcourt beaucoup plus de branches que les autres.

```
! ------------------------------------------------------------------
! Minimization problem - 7 variables, 50 constraints
! TimeLimit            = 0,1
! LogVerbosity         = Verbose
! SearchType           = IterativeDiving
! Initial process time : 0,00s (0,00s extraction + 0,00s propagation)
!  . Log search space  : 16,3 (before), 16,3 (after)
!  . Memory usage      : 299,5 kB (before), 299,5 kB (after)
! Using parallel search with 4 workers.
! ------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
                      0          7                -
+ New bound is 2
                      0          7    1   F        -
+ New bound is 5
                      2          7    1   F     1 != even(1)
+ New bound is 6
*             9        8  0,03s        1      (gap is 33,33%)
*             7       47  0,03s        1      (gap is 14,29%)
*             6      143  0,03s        1      (gap is 0,00%)
! ------------------------------------------------------------------
! Search completed, 3 solutions found.
! Best objective         : 6 (optimal - effective tol. is 0)
! Best bound             : 6
! ------------------------------------------------------------------
! Number of branches     : 3 370
! Number of fails        : 1 780
! Total memory usage     : 2,7 MB (2,2 MB CP Optimizer + 0,5 MB Concert)
! Time spent in solve    : 0,04s (0,03s engine + 0,00s extraction)
! Search speed (br. / s) : 99 117,6
! ------------------------------------------------------------------
```

Figure 5 – iterativeDiving

```
! ------------------------------------------------------------------
! Minimization problem - 7 variables, 50 constraints
! TimeLimit            = 0,1
! LogVerbosity         = Verbose
! SearchType           = Restart
! Initial process time : 0,01s (0,01s extraction + 0,00s propagation)
!  . Log search space  : 16,3 (before), 16,3 (after)
!  . Memory usage      : 299,5 kB (before), 299,5 kB (after)
! Using parallel search with 4 workers.
! ------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
                      0          7                -
+ New bound is 2
                      0          7    1   F        -
+ New bound is 5
                      2          7    1   F     1 != even(1)
+ New bound is 6
*             9        8  0,02s        1      (gap is 33,33%)
*             7       47  0,02s        1      (gap is 14,29%)
*             6      143  0,02s        1      (gap is 0,00%)
! ------------------------------------------------------------------
! Search completed, 3 solutions found.
! Best objective         : 6 (optimal - effective tol. is 0)
! Best bound             : 6
! ------------------------------------------------------------------
! Number of branches     : 481
! Number of fails        : 82
! Total memory usage     : 2,7 MB (2,2 MB CP Optimizer + 0,5 MB Concert)
! Time spent in solve    : 0,02s (0,01s engine + 0,01s extraction)
! Search speed (br. / s) : 30 062,5
! ------------------------------------------------------------------
```

Figure 6 – Restart

1.4 SearchPhase

Les options de searchPhase permettent de choisir l'ordre dans lequel seront sélectionnées les variables et les valeurs. On reprend le modèle naïf et on compare différentes méthodes de sélection. Dans un premier temps, on compare différentes méthodes de sélection des variables. Les valeurs sont sélectionnées dans l'ordre croissant et seules la sélection des variables varie. On observe dans le Tableau 1 que le solveur parcourt autant de branches lorsqu'on sélectionne la variable de plus grand domaine ou de plus petit domaine ou si on les sélectionne par indices croissant. Cela est sans doute dû au fait que les domaines sont très petits ici. Cela reste cependant moins de branches parcourues qu'avec le comportement par défaut.

Méthode de sélection	LargestDomainSize	SmallestDomainSize	Dans l'ordre des indices	Sans phase
Nombre de branches	2245	2246	2246	2499

Table 1 – Selection des variables

Dans un deuxième temps, on fixe la méthode de sélection des variables dans l'ordre croissant de leur indice et on fait varier la méthode de sélection des valeurs. On observe dans le Tableau 2 que c'est l'option ValueImpact qui a le plus d'influence. Lorsqu'on sélectionne la valeur de moindre impact, on parcourt 2258 branches avant de renvoyer la solution, tandis que lorsqu'on sélectionne la valeur de plus grand impact, on parcourt beaucoup plus de branches avant de trouver la solution (3070).

Méthode de sélection	Dans l'ordre des valeurs	SmallestValueImpact	Sans phase	LargestValueImpact
Nombre de branches	2246	2258	2499	3070

Table 2 – Selection des valeurs

1.5 Propagation des contraintes

Le paramètre de propagation des contraintes permet de décider à quel point on veut maintenir l'arc consistence. Un niveau bas d'arc consistance permettra de gagner du temps en chaque noeud mais peut parcourir plus de noeuds avant de réduire les domaines. Avec les deux modèles, le paramètre de propagation des contraintes n'a pas d'influence sur le comportement du solveur.

2 Cavaliers

Dans ce problème, il s'agit de placer un nombre minimum de cavaliers sur un échiquier de sorte que toutes les cases soient contrôlées (avec les règles classiques des échecs). On note par la suite d la dimension du plateau, qui comporte d^2 cases, couvertes par n cavaliers.

2.1 Modélisation

Le premier modèle propose d'attribuer deux coordonnées à chaque cavalier.

Premier modèle

- Variables :
  $x_1, ..., x_n \in \{1, ..., d\}$
  $y_1, ..., y_n \in \{1, ..., d\}$
- Contraintes :
  - $\forall\, i, j \in \{1, ..., d\}$
    $\sum_{k=1}^{n} (i = x_k \; and \; j = y_k) \; or \; ((|x_k - i| + |y_k - j| = 3) \; and \; (i \neq x_k \; and \; j \neq y_k)) \geqslant 1$

Cette modélisation est concise mais ne permet pas de minimiser avec le mot-clé `minimize`. Cependant on peut réutiliser la même méthode que pour l'allocation de fréquences et décrémenter n jusqu'à ne plus pouvoir trouver de solution.
Le second modèle permet aux cavaliers de se trouver en $(0, 0)$ pour modéliser des cavaliers non placés. De cette manière, on peut maximiser le nombre de cavaliers qui sont effectivement placés en $(0, 0)$ et obtenir notre minimisation par `minimize`.

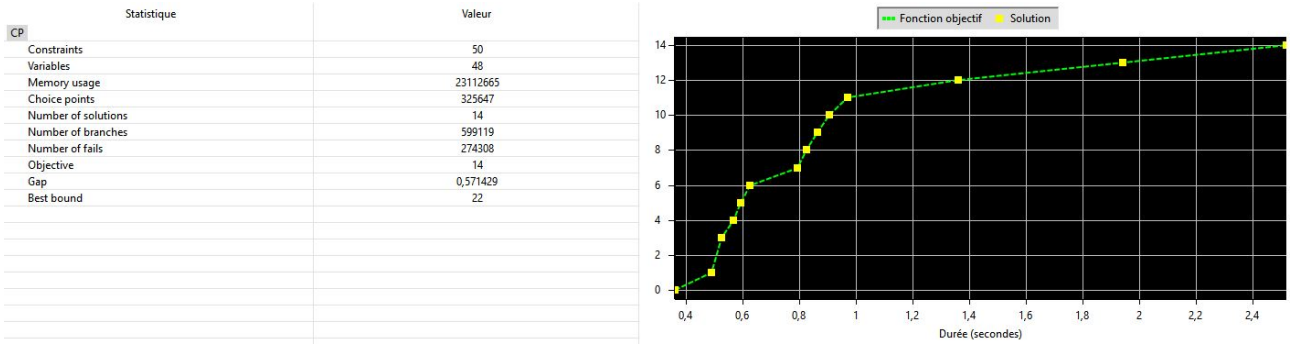
Second modèle

- Variables :
  $x_1, ..., x_n \in \{0, ..., d\}$
  $y_1, ..., y_n \in \{0, ..., d\}$
- Contraintes :
  - $\forall\, i, j \in \{1, ..., d\}$
    $\sum_{k=1}^{n} ((i = x_k \, and \, j = y_k) \; or \; ((|x_k - i| + |y_k - j| = 3) \; and \; (i \neq x_k \; and \; j \neq y_k))) \; and \; (x_k \neq 0) \; and \; (y_k \neq 0) \geqslant 1$
- Objectif : $\max |\{(x_k, y_k) = (0, 0),\; k \in \{1, ..., n\}\}|$

Ces deux modèles ont le même problème en minimisation. Lorsque le premier modèle atteint la valeur à partir de laquelle il n'est plus possible de résoudre les contraintes, il se montre incapable de trouver une borne inférieure lui permettant d'arrêter de parcours de l'arbre de décisions, et le temps mis à montrer qu'il n'existe pas de solution est rapidement déraisonnable. De la même manière, le second modèle ne s'arrête pas lorsqu'il a trouvé l'optimum. Nous les utilisons donc avec un temps limite de 5 secondes.
Pour ce temps, on a qu'à $d = 7$, l'optimum trouvé pour le premier est de 11, alors que le second trouve une solution à 10 cavaliers. Il semblerait que le second soit meilleur, dans la mesure où il exploite `minimize`.

2.2 Type de recherche

On peut s'attendre à ce que les paramètres de recherche aient une grande influence, puisque la vitesse à laquelle l'algorithme va trouver les solutions va être directement déterminante étant donné qu'on estime qu'après 5 secondes, il n'existe plus de solution.
Pour les deux modèles, `auto` et `Restart` ont les meilleurs résultats. Pour $d = 7$, ils trouvent l'optimum à 10 et 11. L'approche `Multipoint` permet au modèle 2 de trouver 15, tandis que le premier trouve 12. `iterative diving` trouve des résultats encore moins bons, et `DepthFirst` ne trouve pas de solution admissible, bien qu'elle explore le plus de branches.



Statistique	Valeur
CP	
Constraints	50
Variables	48
Memory usage	23112665
Choice points	325647
Number of solutions	14
Number of branches	599119
Number of fails	274308
Objective	14
Gap	0,571429
Best bound	22

FIGURE 7 – Statistiques du second modèle $d = 7,\ n = 24$

2.3 SearchPhase

Dans la suite, on n'utilise que le modèle 2.
On va chercher à comparer certaines manières de sélectionner les variables et valeurs :
Pour les variables : `Random`, `SmallestDomain` et `LargestSucessRate`
Pour les valeurs : `Random`, `SmallestValue` et `LargestSucessRate`
Pour $d = 7$, on va observer avec combien de cavaliers l'algorithme arrive à remplir le plateau.
On notera que si on n'arrive pas à trouver de solution admissible, on notera $n = +\infty$.

Successful value	12	$+\infty$	$+\infty$
Smallest value	$+\infty$	$+\infty$	$+\infty$
Random value	12	11	10
$d = 7$	Random variable	Smallest domain variable	Largest success variable

On peut comprendre pourquoi la `SmallestValue` ne facilite pas les choses : il encourage les coordonnées à être instanciées à 0, donc en dehors du plateau. Il semblerait donc que des valeurs aléatoires et des variables de bon success rate soient de bon paramètres. En revanche, si on teste avec $d = 8$, la recherche par défaut trouve $n = 13$ alors que ces paramètres obtiennent $n = 14$.

3 Conclusion

Ce travail a permis de se familiariser avec le solveur de PPC de CPLEX STUDIO. On a pu expérimenter l'utilisation des paramètres de résolution sur différents problèmes et constater les résultats obtenus grâce aux statistiques et journal de moteur intégré dans l'IDE.