Une modélisation LocalSolver pour le placement des assemblages combustibles en piscine

Jean-Yves Lucas[1], Didier Marcel[2],Thierry Benoist[3], Frédéric Gardi[3], Romain Megel[3]

[1] EDF R&D, 1. av. du Général de Gaulle, 92140 Clamart
jean-yves.lucas@edf.fr
[2] ENSTA ParisTech / CNAM, 828, Boulevard des Maréchaux, 91762 Palaiseau Cedex
[3] LocalSolver, 24 Avenue Hoche, 75008 Paris
{tbenoist,fgardi,rmegel}@localsolver.com

Mots-clés *: Quadratic Assignment, LocalSolver*

1 Introduction

Le combustible présent dans le cœur d'un réacteur nucléaire est constitué de nombreuses unités élémentaires appelées « assemblages combustible ». Le nombre de ces assemblages varie selon la puissance nominale du réacteur, d'environ 150 pour les premières générations de réacteurs REP jusqu'à environ 240 pour la plus récente (EPR). Lors d'un cycle de production, chaque assemblage est muni sur sa partie supérieure d'un outillage spécifique appelé grappe, d'un type particulier. Il existe quatre types de grappes. En fonction de considérations neutroniques, chaque assemblage devra être muni d'une grappe du type requis, selon sa position dans le cœur du réacteur. Le combustible nucléaire s'épuisant progressivement au cours des cycles de production, il est nécessaire de prévoir régulièrement (environ une fois par an) un remplacement partiel des assemblages, soit par tiers, soit par quart, lors des arrêts pour maintenance et rechargement. Pour cela l'ensemble des assemblages du cœur sont déchargés du réacteur et placés dans la piscine de refroidissement. Le tiers (ou le quart) des assemblages les plus usés resteront dans la piscine après la fin de l'arrêt, et seront remplacés par autant d'assemblages neufs, déjà présents en piscine avant la phase de déchargement. Afin de munir chaque assemblage participant à la campagne de production suivante de la grappe requise, un robot de manutention met en œuvre une séquence ordonnée de permutations des grappes. A l'issue de celle-ci, chaque assemblage (neuf ou non) entrant en cœur pour le cycle de production suivant aura été muni d'une grappe du bon type, cédée par un assemblage présent en cœur au cycle précédent. Connaissant l'ordre de visite des assemblages par le robot, il convient de disposer ceux-ci dans les alvéoles libres de la piscine de façon à minimiser le temps total du trajet. En effet le temps est précieux lors de ces arrêts: une journée d'indisponibilité coûte environ un million d'euros, d'où l'importance d'optimiser cette tâche qui peut se trouver sur le chemin critique. C'est ce « problème du placement » que nous abordons dans cette étude.

Ce problème de placement d'assemblages en piscine se formalise comme un problème d'affectation quadratique ou QAP. La formulation classique du problème consiste à placer N objets sur P emplacements déterminés ($P \geq N$) pour minimiser une fonction de coût définie en fonction du placement relatif des objets entre eux. Ce problème, largement étudié dans la littérature, est connu pour être NP-complet, et ne pas disposer aujourd'hui de bonnes méthodes de calcul de bornes. Toutefois, notre problème a une propriété intéressante : le coût du placement relatif des

assemblages *i* et *j* n'est défini que si ces assemblages sont visités consécutivement lors du trajet du robot.

2 Modélisation et résolution

Ce problème se prête mal à une modélisation linéaire en nombre entiers. Certaines linéarisations existent dans la littérature mais nous avons pu constater expérimentalement qu'elles étaient inopérantes pour des instances de grandes tailles comme les cas réels que nous considérons ici. Ce constat nous a conforté dans notre souhait de tester une approche par recherche locale à l'aide de LocalSolver. Ce solveur de programmation mathématique est de type *model & run*, c'est-à-dire qu'il prend en entrée une modélisation dans un formalisme mathématique simple et la résolution est ensuite totalement automatique. Par exemple dans le cas présent, les variables et les contraintes s'écrivent en quelques lignes.

```
function model() {
    //variables 0-1: z[a][p] = 1 si l'assemblage a est affecté à l'alvéole p
    z[a in 1..nb_assemblages][p in 1..nb_alveoles] <- bool();

    //affectation: une alvéole par assemblage et au plus un assemblage par alvéole
    for[a in 1..nb_assemblages]  constraint sum[p in 1..nb_alveoles] (z[a][p]) == 1;
    for[p in 1..nb_alveoles]  constraint sum[a in 1..nb_assemblages](z[a][p]) <= 1 ;

    // coordonnées (x[a],y[a]) de l'alvéole affectée à l'assemblage a,
    // connaissant les coordonnées (X[p],Y[p]) des alvéoles (donnée)
    x[a in 1..nb_assemblages] <- sum[p in 1..nb_alveoles](X[p]* z[a][p]);
    y[a in 1..nb_assemblages] <- sum[p in 1..nb_alveoles](Y[p]* z[a][p]);

    // longueur du trajet de S[i-1] vers S[i] où S est la séquence des assemblages
    // c'est à dire le parcours du robot(donnée).
    longueur[i in 1..n] <- max(abs(x[S[i])-x[S[i-1]]), abs(y[S[i])-y[S[i-1]]));

    // objectif
    minimize sum[i in 1..n](longueur[i]);
}
```

L'une des forces de LocalSolver est la faculté qu'a l'utilisateur d'utiliser des expressions numériques quelconques (éventuellement via les fonctions qu'il définit). Dans notre cas par exemple il nous a été possible d'utiliser de manière naturelle les primitives pré-définies max et abs de LocalSolver pour définir la distance de Tchebychev. Ceci est évidemment plus compliqué en modélisation linéaire et impose par exemple l'ajout de nombreuses variables binaires.

Selon les instances, le nombre de grappes à permuter allait de 157 à 241, le nombre d'assemblages (y compris les assemblages neufs) de 209 à 321, le nombre de déplacements élémentaires du robot était compris généralement entre 200 et 300. Ces instances réelles induisent une combinatoire gigantesque, même si trouver une solution au problème reste relativement facile : « il suffit » de placer chaque assemblage dans un emplacement disponible. Néanmoins trouver une (très) bonne solution reste très difficile. Sur les différents jeux de données, nous avons laissé tourner LocalSolver sur des temps allant de quelques secondes à 1 heure. Pour chaque instance, des solutions ont été trouvées dès les premières secondes de la recherche, et ensuite, celles-ci ont été régulièrement améliorées, jusqu'au temps limite d'une heure que nous nous étions fixé. L'amélioration entre les premières solutions fournies et celles trouvées au bout d'une heure est de l'ordre de 3% à 7% selon les cas

Testé sur des instances réelles d'un problème combinatoire complexe, LocalSolver a montré sa capacité à fournir des solutions en des temps compatibles avec les contraintes du processus opérationnel. Des développements sont actuellement en cours pour concevoir des algorithmes heuristiques dédiés au problème et utilisant sa structure. Le choix final de la méthode qui sera retenue pour la mise en production sera fonction de la qualité des solutions, des temps de calculs et de la simplicité du programme.