Table des matières

1 Étude bibliographique

2 A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems[1]

Objectif de l'article : Présenter différents algorithmes, constructifs ou non, ainsi que plusieurs heuristiques en vue de résoudre le TDC (expliqué après).

Problème considéré : *Two-dimensional cutting problem* = **une** grande feuille de papier et un ensemble de pièces **rectangulaires** à découper. Chaque pièce a une **valeur** et il s'agit de **maximiser** la valeur totale découpée. À la différence avec notre problème, il n'y a **ni défauts, ni contrainte de tailles, ni contraintes d'ordre**... Le problème est donc différent. De plus, les différentes pièces ont encore une fois des offres et des demandes différentes. Toutefois, on dispose des **contraintes guillotines** (notons que les pièces ne peuvent pas roter ici) (la rotation semble être une liberté très importante).

Les TDC se classent en quatre catégories :

1. *The unconstrained unweighted version* (UU_TDC) : la valeur d'une pièce est égale à sa surface. On cherche à maximiser la surface utilisée (ce que nous on veut, d'une certaine manière).
2. *The unconstrained weighted version* (UW_TDC) : Chaque pièce a une certaine valeur, indépendante de sa surface (a priori). Pas ce qu'on veut.
3. *The constrained unweighted version* (CU_TDC) : On contraint sur les quantités à produire de chaque pièce. Il s'agit encore plus de notre problème puis-ce qu'on impose ici de couper 1 fois chaque pièce.
4. *The constrained weighted version*

— Des algorithmes exacts existent pour des petites et moyennes instances. Pour les autres, il y a des heuristiques : l'une d'elle se base sur une recherche en profondeur d'abord suivie d'une *hill climbing strategy* (algorithme DH)

A constructive heuristic algorithm : L'idée est de considérer, pour chaque sous-rectangle à découper toutes les pièces possibles (ainsi que ses orientations) :
Pour une pièce placée dans le coin inférieur gauche, une guillotine va faire deux nouvelles pièces. On calcule pour chacune d'entre elle une borne supérieure de ce qu'on pourrait y mettre en résolvant un problème du sac-à-dos. L'idée étant de faire en sorte de pouvoir en mettre un maximum.
Utilisation de l'algorithme *fast greedy algorithm* de Martello et Toth pour obtenir cette valeur (suffisamment décrit dans l'article).
L'autre borne sup. proposée dans l'article ne s'applique pas à notre problème puis-ce qu'on ne découpe pas plusieurs fois une pièce.
Cet algorithme revient à notre algorithme glouton.

A GRASP algorithm : GRASP pour *greedy adaptative search procedure* (greedy = glouton). À chaque itération de GRASP il y a une partie constructive et une procédure d'échange (recherche locale).
L'algorithme se structure comme tel :

1. On prend un rectangle courant (le plus petit ou le premier, pour vérifier la contrainte d'ordre dans notre cas).
2. Pour chaque pièce qui rentre on calcule un score intelligent (le max des bornes sup + la valeur de la pièce). Score max noté B.
3. On prend aléatoirement une pièce possible parmi toutes celles qui ont un score $\geq \delta.B$ où δ est un paramètre d'acceptation (=> on rajoute de l'aléa pour parcourir plus de solutions).
4. Ensuite, on fusionne les pièces déchets avec les pièces adjacentes dans l'optique de créer un grand rectangle qui puisse être redécoupé. Des pièces dites adjacentes sont des pièces qui ont un côté (obligatoirement entier ici) en commun c-à-d deux enfants d'un même papa.
5. On réapplique le glouton sur le rectangle fusionné.

Attention : Ça va être la grosse merde pour vérifier la contrainte d'ordre ce truc non ?

A tabu search algorithm :

- Un *move* qui doit vérifier la contrainte guillotine. Ce que eux décrivent : étant donné deux rectangles qui partagent un morceau de côté en commun et considérent le plus petit rectangle qui les contienne tous les deux. Ils déterminent ensuite le plus petit rectangle dont la guillotine cut correspond. Ils vident alors ce rectangle et le redécoupent avec l'algorithme GRASP.
- La *sélection du move* : à chaque itération ils selectionnent uniformément un rectangle (pièces et déchets). Ensuite, pour chaque rectangle voisin ils fusionent pour obtenir un plus grand rectangle (*le move*). On ne fait rien s'il n'y pas de waste sur cette partie.
- On apliquant le move pour chacun de ses voisins, on obtient plusieurs cutting patterns différents. On sélectionne celui qui améliore la fonction objectif.
- La *tabu list* : pour chaque move, ils conservent dans la liste les coordonnées et dimensions des rectangles coupés à nouveau ainsi que le type de pièce dans le coin inférieur gauche.
  Si on améliore pas la solution courante, on change la longueur de la liste aléatoirement.
- La *basic tabu search procedure* : Description générale de l'algorithme.

Remarques : Le voisinage peut-être très général et conduire à refaire entièrement la répartition au sein d'un bin. Une amélioration dans la fonction de décision est de considérer les n meilleures solutions courantes et de changer la valeur des pièces en fonction de leurs apparitions dans ces solutions : plus une pièce apparaît souvent dans les meilleurs solutions, plus on a envie de la garder.

Path relinking : Étant donné deux solutions, on peut passer de l'une à l'autre en effectuant des échanges puis en parcourant ces chemins on arrive à faire de grandes choses...

3 Remarques préliminaires sur le sujet

3.1 Définitions

- Un **jumbo** est une grande fenêtre. Ils sont ordonnés dans un **bin**.
- Une **fenêtre/vitre/item** est une petite fenêtre.
- Les **fenêtres** sont ordonnées au sein de stacks (ordre d'extraction strict) qui eux ne sont pas ordonnés.
- Peut-on commencer par un Tooceuht ?

Références

[1] R. Alvarez-Valdés, A. Parajón, et al. A tabu search algorithm for large-scale guillotine (un) constrained two-dimensional cutting problems. *Computers & Operations Research*, 29(7) :925–947, 2002.