Table des matières

1 Description du problème **2**

2 La solution initiale **3**
1 La modélisation 3
2 L'algorithme 3
3 L'implémentation 3

A Implémentation lpsolve **4**

Partie 1

Description du problème

Le problème de planification de production étudié dans cet exercice est appelé *Uncapacitated Lot-Sizing Problem*. Comme son nom l'indique, il n'y a dans ce cas précis aucune contrainte de capacité ou de limite de production. Le plan de production est décomposé en T instances, dont on cherche à décider l'ouverture ou non (y_t), ainsi que la quantité d'éléments à produire x_t. L'ouverture d'une instance entraîne un coût fixe (f_t), ainsi qu'un coût unitaire de production (p_t). Il faut impérativement subvenir aux demandes des clients pour tout instant t, mais il est possible à une instance t de produire plus que nécessaire, et stocker l'ensemble de la "surproduction" (s_t) jusqu'à $t+1$ moyennant un coût unitaire de h_t.

Nous tentons dans cette analyse de résoudre quelques instances de ce problème via un algorithme de **branch & bound**, et nous essayons de comprendre quelles sont les astuces de modélisation et les choix de résolution permettant d'améliorer au fur et à mesure notre rapidité de résolution.

Partie 2

Le projet initial

1 La modélisation

Les algorithmes de branch & bound sont très utiles pour résoudre des problèmes en variables entières, en effectuant une série de résolution du problème "relaché" et en introduisant à chaque étape une nouvelle contrainte d'intégrité sur une des variables de décision. Nous introduisons donc un modèle correspondant au problème ci-dessus, mais dont les variables entières ou binaires (y_t) ne possèdent pas de contraintes d'intégrité.

Voici le modèle proposé :

$$\left[\begin{array}{lcll} minz = & \sum_{t\in\{1...T\}} y_t \times f_t + x_t \times p_t + s_t \times h_t & & \\ s/c & s_{t-1} + x_t & = d_t + s_t & \forall t \in \{0...T\} \\ & y_t * M & \geq x_t & \\ & s_0 & = 0 & \\ & s_t & = 0 & \\ & y_t, x_t, s_t & \geq 0 & \end{array}\right]$$

La première contrainte, que nous appelons contrainte d'équilibre, impose l'absence de perte de produit lors de la production. En effet, l'ensemble des éléments entrants (stock précédent et production) soit être égal à l'ensemble des éléments sortants (ventes et stock). La seconde contrainte implémente une constante M très grande imposant l'ouverture de l'instance si au moins un élément est produit par cette instance. Les contraintes (3) et (4) indiquent l'état du stock au départ et à l'arrivé, fixés ici à 0.

2 L'algorithme

Pour résoudre entièrement le problème, nous avons exécuté à la main l'algorithme du branch & bound en effectuant les choix suivants :

- Relâcher ou contraindre les variables y_t à 0 ou 1.
- Toujours choisir la variable y_i la plus proche de 0 et la fixer à 0 (dans un premier temps)

3 L'implémentation

Nous utilisons pour la résolution le solveur GLPK sous sa forme "lpsolve". L'implémentation de la modélisation dans ce langage est donné en Annexe A Afin de contraindre une variable, nous ajoutons à chaque étape une ligne de type :

```
s.t. contrX :y[A]=B;
```

avec A=t et B$\in \{0, 1\}$

4 Résultats

Annexe A

Implémentation lpsolve

```
 #Données
param nbpostes;

set T:=1..nbpostes;
set T2:=0..nbpostes;
param M:=25;

param d{T};
param h{T};
param p{T};
param f{T};

#Variables de décision
var x{T} >=0;
var y{T} >=0;
var s{T2}>=0;

#Fonction objectif
minimize couts : sum{i in T} (y[i]*f[i] + x[i]*p[i] + h[i]*s[i]);

#Contraintes

s.t. Equilibre{i in T} : (s[i-1]+x[i])=(s[i]+d[i]);
s.t. Activation{i in T}: y[i]*sum{j in i..nbpostes}d[j] >= x[i];
s.t. contr1 :s[0]=0;
s.t. contr2 :s[nbpostes]=0;

#Résolution
solve;

#Affichange des res
display : s;
display : x;
display : y;
display : sum{i in T} (y[i]*f[i] + x[i]*p[i] + h[i]*s[i]); #fonction objectif à recopier

data;

param nbpostes:= 5;
param d:= 1 3 2 5 3 6 4 3 5 8;
```

```
param p:= 1 2 2 4 3 6 4 8 5 10 ;
param h:= 1 3 2 2 3 3 4 2 5 0;
param f:= 1 10 2 8 3 6 4 4 5 2;

end;
```