Projet : Implémentation d'un algorithme de type *Branch & Bound* appliqué à la résolution de problème de type *lot sizing*

Lardenois Adrien — Mahéo Arthur

M1 ORO
Premier semestre 2012

Table des matières

1 Préambule **1**

2 Multi-Item Capacitated Lot Sizing (MCLS) **2**
- 2.1 Problème . . . 2
  - 2.1.1 Notions . . . 2
  - 2.1.2 Différence entre ressources et coûts . . . 3
- 2.2 Programme linéaire . . . 3
  - 2.2.1 Contrainte . . . 3
  - 2.2.2 Modèle de base avec retards . . . 3
- 2.3 Instance réduite . . . 4
  - 2.3.1 Données . . . 4
  - 2.3.2 Variables . . . 4
  - 2.3.3 Fonction objectif . . . 4
  - 2.3.4 Contraintes . . . 5
  - 2.3.5 Grand M . . . 5
  - 2.3.6 Heuristique . . . 5
  - 2.3.7 Pseudo-contrainte . . . 5
- 2.4 Implémentation . . . 6
  - 2.4.1 Données . . . 6
  - 2.4.2 Matrice creuse . . . 6
- 2.5 Résultats . . . 6
  - 2.5.1 Taille 2 x 2 . . . 6
  - 2.5.2 Taille 3 x 3 . . . 8
  - 2.5.3 Taille 4 x 4 . . . 8
  - 2.5.4 Taille 5 x 5 . . . 9
  - 2.5.5 Taille 6 x 6 . . . 9

3 *Branch and Cut* **10**
- 3.1 Coupe . . . 10
- 3.2 Résultats . . . 10
  - 3.2.1 Taille 2 x 2 . . . 10
  - 3.2.2 Taille 3 x 3 . . . 11
  - 3.2.3 Taille 4 x 4 . . . 13
  - 3.2.4 Taille 5 x 5 . . . 13

Chapitre 1

Préambule

Les algorithme *Branch & Bound* s'appliquent à des problème en variables entières. Ils se reposent sur la relaxation linéaire de variables entières pour borner la solution recherchée et se déroule en deux grandes étapes :

1. la recherche d'une solution admissible ;
2. l'amélioration de cette solution optimale en explorant divers combinaisons de variables.

Tant que l'on a pas de solution admissible, l'algorithme va descendre en serrant les variables relâchées une à une en fonction d'une heuristique choisie, puis, une fois cette solution trouvée, on va pouvoir éliminer des branches entières de l'abre. Une branche peut être éliminée pour deux raisons :

1. la combinaison serrée rend le problème insoluble ;
2. la solution du problème relâché est moins intéressante que notre solution optimale actuelle.

Chapitre 2

Multi-Item Capacitated Lot Sizing (MCLS)

Cette variante du problème de *lot sizing*, que nous allons étudier sous une forme réduite, prend en compte le fait que nous avons à produire plusieurs types de produits différents.

2.1 Problème

Le MCLS sert à modéliser la production de produits par une usine. Il est l'évolution d'un problème de type CLS pour intégrer le fait que, bien souvent, représenter la production d'un seul produit ne convient pas dans une situation réelle. Pour faire le lien avec ce dernier, en peut, grossièrement, considérer un MCLS comme une combinaison de CLS, chacun des CLS représentant une ligne de production.

Passer à plusieurs produits oblige aussi à revoir certaines données, ainsi le coût de production, de stockage, d'ouverture de ligne, etc. peut devenir fonction du produit et de la période.

Nous pourrions aussi discuter de la possibilité d'avoir un ensemble de lignes de production. Cela ouvre des possibilités diverses telle : production sur plusieurs lignes d'un même produit, restriction de certaines lignes à certains produits, coûts de production et d'activation liées à la ligne, au produit et à la période, etc.

De même, il est possible de modéliser le fait qu'un produit nécessite plusieurs types de ressources différents, avec chacun leurs quantités disponible, voire leurs coûts.

2.1.1 Notions

Backlogs

Les *backlogs*, ou retards, représentent le fait qu'il est possible de ne pas pouvoir satisfaire la demande pour une période donnée. De fait, il leur sera associé un coût spécifique, et une variable pour représenter la quantité de retard prise.

Big and small bucket

Les problèmes de type *big bucket* partent du principe que, pour une période donnée, on peut produire autant de produits différents que l'on veut (sous réserve des autres contraintes).

De l'autre côté les problèmes *small bucket* assument que l'on ne peut produire qu'un seul produit pendant une period donnée.

2.1.2 Différence entre ressources et coûts

Il convient de bien faire la distinction entre les ressources et les coûts, ces deux notions, bien que proche, n'ont pas du tout le même rôle dans la modélisation de notre problème.

Les ressources

Sont en quantité limitées et sont consommées lors de la production, voire de l'ouverture d'une ligne de produits. Les capacités C_t^i peuvent être décrites telles les ressources disponibles.

Les coûts

De même, les coûts peuvent être liés aux produits et aux périodes, néanmoins ils se répercutent dans la fonction objectif, ce sont eux que nous cherchons à minimiser. Pour faire le lien avec une situation réelle, les coûts d'un produit incluent le coût des ressources qu'il nécessite.

2.2 Programme linéaire

2.2.1 Contrainte

$$\sum_{i=1}^{N}(v_t^i x_t^i + \phi_t^i y_t^i) \leq C_t, \forall t \tag{2.1}$$

Le membre gauche de cette contrainte représente le coût en ressources de nos produits tant en production v_t^i qu'en activation de ligne ϕ_t^i.

Le membre droit représente la capacité dont nous disposons par période, autrement dit les ressources disponibles.

2.2.2 Modèle de base avec retards

$$min\ z = \sum_{i=1}^{N}\sum_{t=1}^{T}(p_t^i x_t^i + h_t^i s_t^i + b_t^i r_t^i + f_t^i y_t^i) \tag{2.2}$$

T le nombre de périodes ;
N le nombre de produits ;
p_t^i coût de production d'un produit ;
h_t^i coût de stockage d'un produit i en périod t ;
b_t^i coût de retard par produit i ;
f_t^i coût de l'ouverture d'une ligne de production pour un produit i en t ;
ϕ_t^i coût d'activation, en ressources de la production pour un produit ;
v_t^i coût de production, en ressources, d'un produit.
C_t^i capacité de production de i durant la période t.
d_t^i La demande en produit i pendant la période t.

$$\begin{aligned}
x_t^i + s_{t-1}^i + s_t^i &= d_t^i + s_t^i + r_{t-1}^i && \forall i, \forall t \quad (1)\\
x_t^i &\leq M y_t^i && \forall i, \forall t \quad (2)\\
\sum_{i=1}^{N}(v_t^i x_t^i + \phi_t^i y_t^i) &\leq C_t && \forall t \quad (3)\\
x, s, r \in \mathbb{N}, y \in \mathbb{B}
\end{aligned}$$

1. On doit satisfaire la demande d_t^i pour tous les produits pour toutes les périodes. En d'autres termes il nous faut équilibrer la production et la demande. Or la production seule ne suffit pas à représenter la satisfaction ou non de la demande, il faut aussi prendre en compte le stock de la période précédente (en ajout), ainsi que la mise en stock durant la période en cours (en retrait). De plus nous avons le droit à un retard (en retrait) mais il nous faut satisfaire le retard pris en période précédente (en ajout). Pour conclure, on cherche à équilibrer les disponibilités avec les consommations.

2. Il faut activer une ligne à partir du moment où l'on produit dessus. Autrement dit, il faut payer le coût d'ouverture de la ligne pour toute production lancée.
3. On ne peut dépasser, en production, la capacité totale par période. Autrement dit, on ne peut produire plus que ce que les ressources disponibles permettent.

2.3 Instance réduite

Nous travaillerons sur une instance réduite du problème présentant ces variations :
- Les coût de production sont nuls $p_t^i = 0$.
- Les coûts de lancement de production sont nuls $\phi_t^i = 0$.
- La capacité ne se basent que sur les périodes C_t.
- Les coûts de production sont unitaires $v_t^i = 1$.
- Les coûts d'ouverture sont par produit f^i.
- Les coûts de stockage sont par produit h^i.
- Les coûts de retard sont par produit b^i.

2.3.1 Données

T le nombre de périodes ;
N le nombre de produits ;
h^i coût de stockage d'un produit i.
b^i coût de retard par produit i ;
f^i coût de l'ouverture d'une ligne de production pour un produit i ;
C_t capacité de production durant la période t.
d_t^i La demande en produit i pendant la période t.

2.3.2 Variables

Notre modèle comprendra quatre variables de décisions :

x_t^i Quantité de produit i produite pendant la période t.

s_t^i Quantité de produit i mise en stock en période t.

r_t^i Quantité de produit i non-délivrée, donc en retard, en période t.

y_t^i Production de i en période t.

2.3.3 Fonction objectif

$$min\ z = \sum_{i=1}^{N} \sum_{t=1}^{T} (h_t^i s_t^i + b_t^i r_t^i + f_t^i y_t^i) \tag{2.3}$$

2.3.4 Contraintes

1. On doit satisfaire la demande d_t^i pour tous les produits pour toutes les périodes. En d'autres termes il nous faut équilibrer la production et la demande. Or la production seule ne suffit pas à représenter la satisfaction ou non de la demande, il faut aussi prendre en compte le stock de la période précédente (en ajout), ainsi que la mise en stock durant la période en cours (en retrait). De plus nous avons le droit à un retard (en retrait) mais il nous faut satisfaire le retard pris en période précédente (en ajout).
2. Il faut activer une ligne à partir du moment où l'on produit dessus. Autrement dit, il faut payer le coût d'ouverture de la ligne pour tout produit en production.

3. On ne peut dépasser, en production, la capacité totale par période, et ce en prenant bien en compte le fait qu'ouvrir une ligne pour un produit ne coûte pas de ressources et que le coût, en ressources toujours, de production d'un produit est unitaire.

Ce qui nous donne le modèle suivant, mis en relief pour GLPK avec les limites à droite et les coefficients mis en valeur.

$$\begin{aligned} x_t^i + s_{t-1}^i - s_t^i - r_{t-1}^i + s_t^i &= d_t^i \quad \forall i, \forall t \quad (1) \\ x_t^i - M y_t^i &\leq 0 \quad \forall i, \forall t \quad (2) \\ \sum_{i=1}^{N} x_t^i &\leq C_t \quad \forall t \quad (3) \\ x, s, r \in \mathbb{N}, y \in \mathbb{B} & \end{aligned}$$

2.3.5 Grand M

La méthode du « grand M » sert à imposer une valeur à une variable entière par rapport à une variable binaire : on veut soit les deux égales à 0, soit la variable binaire à 1 tandis que la seconde peut prendre la valeur qu'elle souhaite. Le choix de la valeur de M peut impacter sérieusement la rapidité de notre algorithme en permettant une élimination plus rapide des branches in-intéressantes. On recherche une valeur suffisamment grande pour ne pas limiter la progression de la variable entière tout en restant suffisamment proche de sa valeur maximale pour optimiser l'élimination des branches.

Dans notre cas deux possibilités assez évidentes s'imposent pour le définir :

1. Nous posons $M_t = C_t$. Dans tous les cas nous ne pourrons pas outrepasser notre capacité de production.
2. Nous posons $M_t^i = \sum_{t'=t}^{T} d_{t'}^i$. Notre valeur est la somme des demandes à venir pour un produit, car nous n'aurons pas besoin, au final, de produire plus que ça.

Dans un cas comme dans l'autre nous pouvons trouver des exemples permettant de les remettre en question :

1. Notre capacité de production dépasse de très loin les demandes par période.
2. Nous avons des demandes très proches des capacités de production par période, donc les premières valeurs seront assez peu limitantes.

Finalement, avec deux valeurs ayant chacune leurs avantages, nous pouvons nous orienter vers : 3. $min(M_t, M_t^i)$.

2.3.6 Heuristique

Un autre point important dans la modélisation du problème est notre heuristique pour choisir la première valeur des y_t^i. Ici nous sélectionnerons le y_t^i le plus petit et commencerons par explorer la branche le fixant à zéro.

2.3.7 Pseudo-contrainte

Nous considérons que le plan de production doit être rempli au mieux, et imposons donc qu'il n'y ait pas de retard ni de stock en dernière période.

$$s_T^i + r_T^i = 0 \quad (2.4)$$

2.4 Implémentation

L'implémentation se fera en C++, en utilisant GLPK en tant que librairie. Nous utiliserons un modèle récursif pour le parcours des solutions. Nous utiliserons une structure pour sauvegarder notre meilleure solution. Puis à chaque étape :

- on résout le problème ;
- si le problème est réalisable ($z > 0$) et intéressant :
  - si la solution est admissible (toutes les y_t^i sont serrées), alors elle notre nouvelle meilleure ;
  - sinon on la contraint à 0 puis à 1 en rappelant la résolution entre chaque ;

Pour fixer nos y_t^i nous nous contentons d'ajouter une contrainte au problème.

$$y_t^i = 0 \; ou \; y_t^i = 1 \quad (2.5)$$

2.4.1 Données

Nous définissons nos quatre variables en mettant le coefficient des x_t^i à 0. Puis nos trois contraintes avec leurs membres droits.

2.4.2 Matrice creuse

Le point principal de la librairie GLPK est la matrice des contraintes. Au vu de la taille des données, nous utilisons une représentation par matrice creuse. Par une double boucle sur les périods puis les produits nous allons définir les coefficients par variable pour chacune de nos contraintes. Soit $\Gamma = N \times T$, nous avons trois jeux de Γ variables, et trois jeux de Γ contraintes — que nous allons affiner par la suite.

Pour réduire la taille de notre matrice creuse, nous éliminons tout d'abord les références à s_{t-1}^i et r_{t-1}^i lorsque $t = 1$. De même, du fait de notre pseudo-contrainte, on élimine les s_t^i et r_t^i lorsque $t = T$.

Pour définir $\sum_{t=1}^{T} x_t^i$, il nous suffit de baser leur indice de contrainte uniquement sur t.

Notre matrice creuse a donc une taille de : $8\Gamma - 4T + 1$

2.5 Résultats

2.5.1 Taille 2 x 2

Taille maximale de l'arbre : 31 nœuds. Trouvé : 340 en : 15 itérations.

$$\begin{array}{ll} x_1^1 = 0 & x_1^2 = 0 \\ x_2^1 = 70 & x_2^2 = 60 \end{array}$$

$$\begin{array}{ll} s_1^1 = 0 & s_1^2 = 0 \\ s_2^1 = 0 & s_2^2 = 0 \end{array}$$

$$\begin{array}{ll} r_1^1 = 0 & r_1^2 = 20 \\ r_2^1 = 0 & r_2^2 = 0 \end{array}$$

$$\begin{array}{ll} y_1^1 = 0 & y_1^2 = 0 \\ y_2^1 = 1 & y_2^2 = 1 \end{array}$$

2.5.2 Taille 3 x 3

Taille maximale de l'arbre : 1023 nœuds. Trouvé $z = 690$ en 137 itérations.

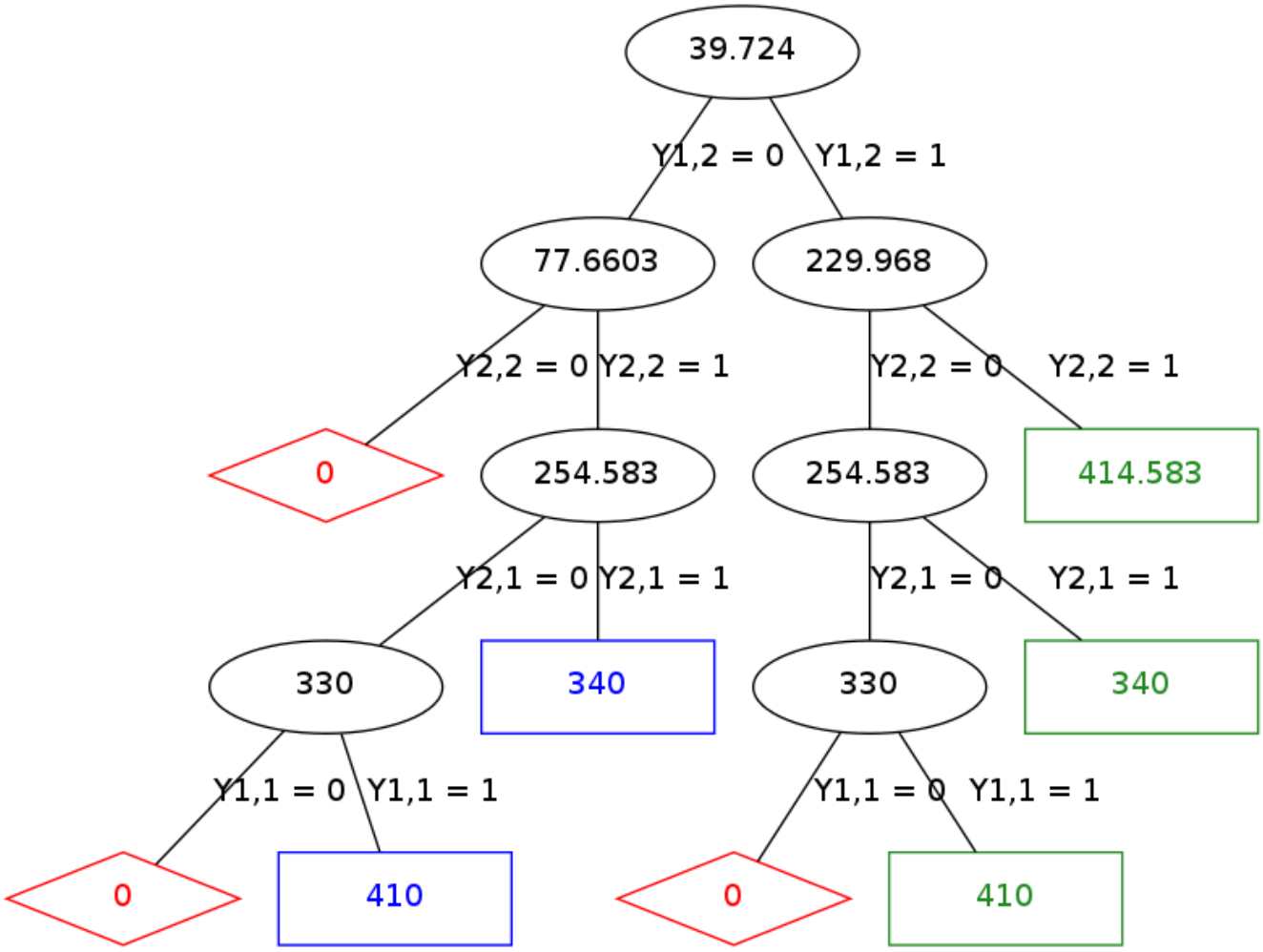
$$\begin{array}{lll} x_1^1 = 0 & x_1^2 = 0 & x_1^3 = 90 \\ x_2^1 = 120 & x_2^2 = 110 & x_2^3 = 0 \\ x_3^1 = 0 & x_3^2 = 0 & x_3^3 = 0 \end{array}$$

$$\begin{array}{lll} s_1^1 = 0 & s_1^2 = 0 & s_1^3 = 50 \\ s_2^1 = 50 & s_2^2 = 50 & s_2^3 = 0 \\ s_3^1 = 0 & s_3^2 = 0 & s_3^3 = 0 \end{array}$$

$$\begin{array}{lll} r_1^1 = 0 & r_1^2 = 20 & r_1^3 = 0 \\ r_2^1 = 0 & r_2^2 = 0 & r_2^3 = 0 \\ r_3^1 = 0 & r_3^2 = 0 & r_3^3 = 0 \end{array}$$

$$\begin{array}{lll} y_1^1 = 0 & y_1^2 = 0 & y_1^3 = 1 \\ y_2^1 = 1 & y_2^2 = 1 & y_2^3 = 0 \\ y_3^1 = 0 & y_3^2 = 0 & y_3^3 = 0 \end{array}$$

FIGURE 2.1 – B&B 2x2



L'arborescence de recherche dépasse la taille possible pour la représentation.

2.5.3 Taille 4 x 4

Taille maximale de l'arbre : 131 071 nœuds. Trouvé $z = 1660$ en 7839 itérations.

$$
\begin{array}{llll}
x_1^1 = 0 & x_1^2 = 0 & x_1^3 = 90 & x_1^4 = 0 \\
x_2^1 = 120 & x_2^2 = 120 & x_2^3 = 0 & x_2^4 = 0 \\
x_3^1 = 0 & x_3^2 = 0 & x_3^3 = 0 & x_3^4 = 350 \\
x_4^1 = 100 & x_4^2 = 0 & x_4^3 = 100 & x_4^4 = 0
\end{array}
$$

$$
\begin{array}{llll}
s_1^1 = 0 & s_1^2 = 0 & s_1^3 = 50 & s_1^4 = 0 \\
s_2^1 = 50 & s_2^2 = 60 & s_2^3 = 0 & s_2^4 = 0 \\
s_3^1 = 0 & s_3^2 = 10 & s_3^3 = 0 & s_3^4 = 150 \\
s_4^1 = 0 & s_4^2 = 0 & s_4^3 = 0 & s_4^4 = 0
\end{array}
$$

$$
\begin{array}{llll}
r_1^1 = 0 & r_1^2 = 20 & r_1^3 = 0 & r_1^4 = 0 \\
r_2^1 = 0 & r_2^2 = 0 & r_2^3 = 0 & r_2^4 = 100 \\
r_3^1 = 0 & r_3^2 = 0 & r_3^3 = 0 & r_3^4 = 0 \\
r_4^1 = 0 & r_4^2 = 0 & r_4^3 = 0 & r_4^4 = 0
\end{array}
$$

$$
\begin{array}{llll}
y_1^1 = 0 & y_1^2 = 0 & y_1^3 = 1 & y_1^4 = 0 \\
y_2^1 = 1 & y_2^2 = 1 & y_2^3 = 0 & y_2^4 = 0 \\
y_3^1 = 0 & y_3^2 = 0 & y_3^3 = 0 & y_3^4 = 1 \\
y_4^1 = 1 & y_4^2 = 0 & y_4^3 = 1 & y_4^4 = 0
\end{array}
$$

2.5.4 Taille 5 x 5

Taille maximale de l'arbre : 67 108 863 nœuds. Trouvé $z = 2710$ en 764657 itérations.

$$
\begin{array}{lllll}
x_1^1 = 0 & x_1^2 = 0 & x_1^3 = 90 & x_1^4 = 0 & x_1^5 = 120 \\
x_2^1 = 120 & x_2^2 = 150 & x_2^3 = 0 & x_2^4 = 0 & x_2^5 = 0 \\
x_3^1 = 0 & x_3^2 = 0 & x_3^3 = 0 & x_3^4 = 350 & x_3^5 = 0 \\
x_4^1 = 120 & x_4^2 = 0 & x_4^3 = 140 & x_4^4 = 0 & x_4^5 = 0 \\
x_5^1 = 0 & x_5^2 = 0 & x_5^3 = 0 & x_5^4 = 160 & x_5^5 = 0
\end{array}
$$

$$
\begin{array}{lllll}
s_1^1 = 0 & s_1^2 = 0 & s_1^3 = 50 & s_1^4 = 0 & s_1^5 = 70 \\
s_2^1 = 50 & s_2^2 = 90 & s_2^3 = 0 & s_2^4 = 0 & s_2^5 = 70 \\
s_3^1 = 0 & s_3^2 = 40 & s_3^3 = 0 & s_3^4 = 150 & s_3^5 = 50 \\
s_4^1 = 20 & s_4^2 = 30 & s_4^3 = 40 & s_4^4 = 0 & s_4^5 = 10 \\
s_5^1 = 0 & s_5^2 = 0 & s_5^3 = 0 & s_5^4 = 0 & s_5^5 = 0
\end{array}
$$

$$
\begin{array}{lllll}
r_1^1 = 0 & r_1^2 = 20 & r_1^3 = 0 & r_1^4 = 0 & r_1^5 = 0 \\
r_2^1 = 0 & r_2^2 = 0 & r_2^3 = 0 & r_2^4 = 100 & r_2^5 = 0 \\
r_3^1 = 0 & r_3^2 = 0 & r_3^3 = 0 & r_3^4 = 0 & r_3^5 = 0 \\
r_4^1 = 0 & r_4^2 = 0 & r_4^3 = 0 & r_4^4 = 0 & r_4^5 = 0 \\
r_5^1 = 0 & r_5^2 = 0 & r_5^3 = 0 & r_5^4 = 0 & r_5^5 = 0
\end{array}
$$

$$
\begin{array}{lllll}
y_1^1 = 0 & y_1^2 = 0 & y_1^3 = 1 & y_1^4 = 0 & y_1^5 = 1 \\
y_2^1 = 1 & y_2^2 = 1 & y_2^3 = 0 & y_2^4 = 0 & y_2^5 = 0 \\
y_3^1 = 0 & y_3^2 = 0 & y_3^3 = 0 & y_3^4 = 1 & y_3^5 = 0 \\
y_4^1 = 1 & y_4^2 = 0 & y_4^3 = 1 & y_4^4 = 0 & y_4^5 = 0 \\
y_5^1 = 0 & y_5^2 = 0 & y_5^3 = 0 & y_5^4 = 1 & y_5^5 = 0
\end{array}
$$

2.5.5 Taille 6 x 6

Taille maximale de l'arbre : 137 438 953 471 (137.10^9) nœuds. Temps d'exécution trop long.

Chapitre 3

Branch and Cut

Comme vu lors du premier TP, nous pouvons ajouter des « inégalités valides » à notre modélisation afin d'aider à couper des branches plus tôt dans notre arbre. Une inégalité valide est une contrainte supplémentaire ne changeant en rien la solution optimale, mais permettant d'améliorer la qualité de la solution trouvée lors de la relaxation linéaire.

3.1 Coupe

Nous avions développé une inégalité valide traduisant le fait qu'il ne servait à rien de stocker à une période antérieure si l'on ouvrait la production à la période suivante.

$$s_{t-1} \geq d_t(1 - y_t) \tag{3.1}$$

Pour l'intégrer à notre problème, et surtout conserver sa validité, il nous faut la modifier en ajoutant les retards dans le membre droit de l'inégalité.

$$s^i_{t-1} + r^i_t + d^i_t y^i_t \geq d^i_t \tag{3.2}$$

3.2 Résultats

3.2.1 Taille 2 x 2

Taille maximale de l'arbre : 31 nœuds. Trouvé $z = 340$ en 9 itérations.

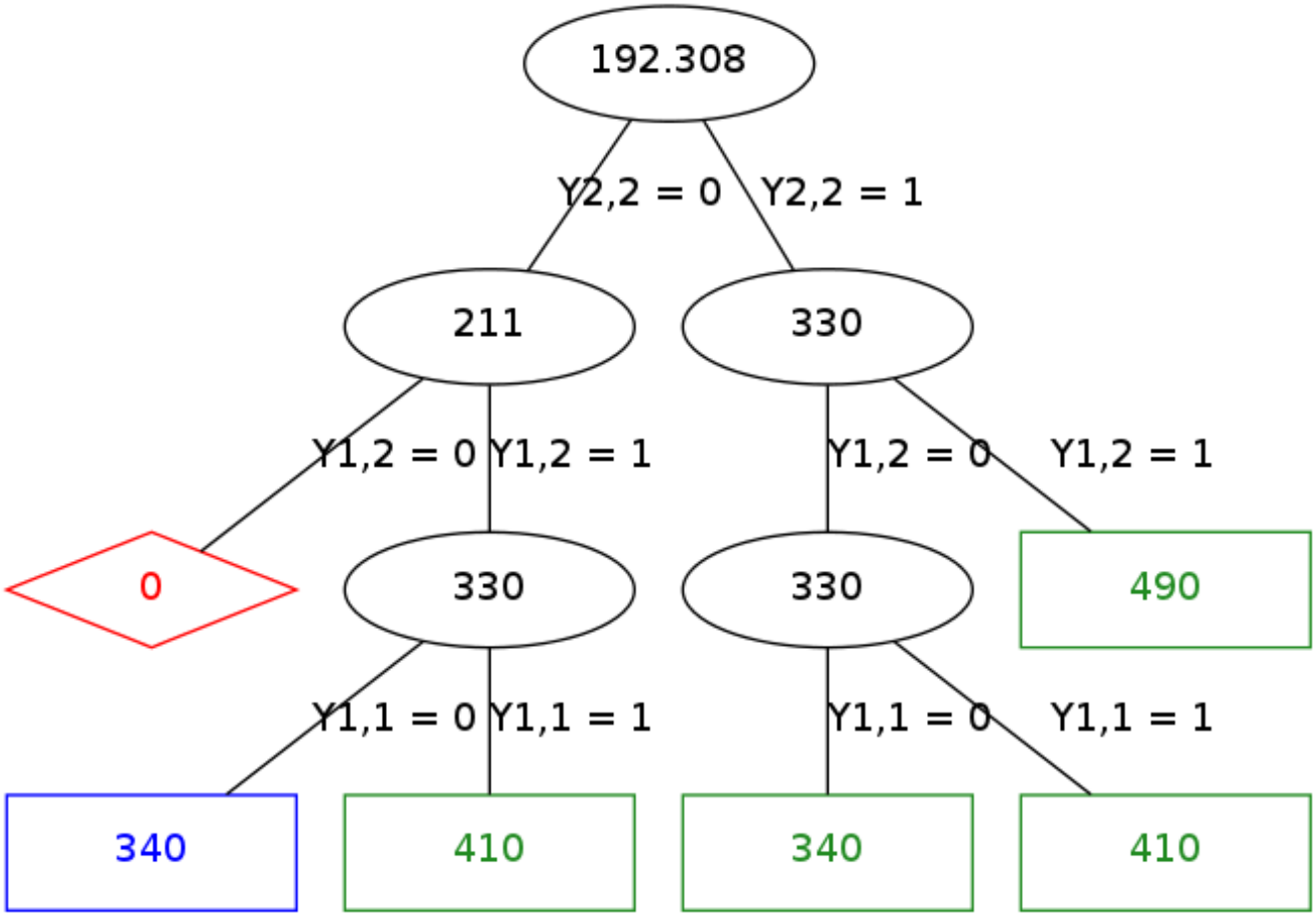
$$\begin{array}{ll} x^1_1 = 0 & x^2_1 = 60 \\ x^1_2 = 70 & x^2_2 = 0 \end{array}$$

$$\begin{array}{ll} s^1_1 = 0 & s^2_1 = 40 \\ s^1_2 = 0 & s^2_2 = 0 \end{array}$$

$$\begin{array}{ll} r^1_1 = 0 & r^2_1 = 0 \\ r^1_2 = 0 & r^2_2 = 0 \end{array}$$

$$\begin{array}{ll} y^1_1 = 0 & y^2_1 = 1 \\ y^1_2 = 1 & y^2_2 = 0 \end{array}$$

FIGURE 3.1 – B&C 2x2



3.2.2 Taille 3 x 3

Taille maximale de l'arbre : 1023 nœuds. Trouvé $z = 690$ en 43 itérations.

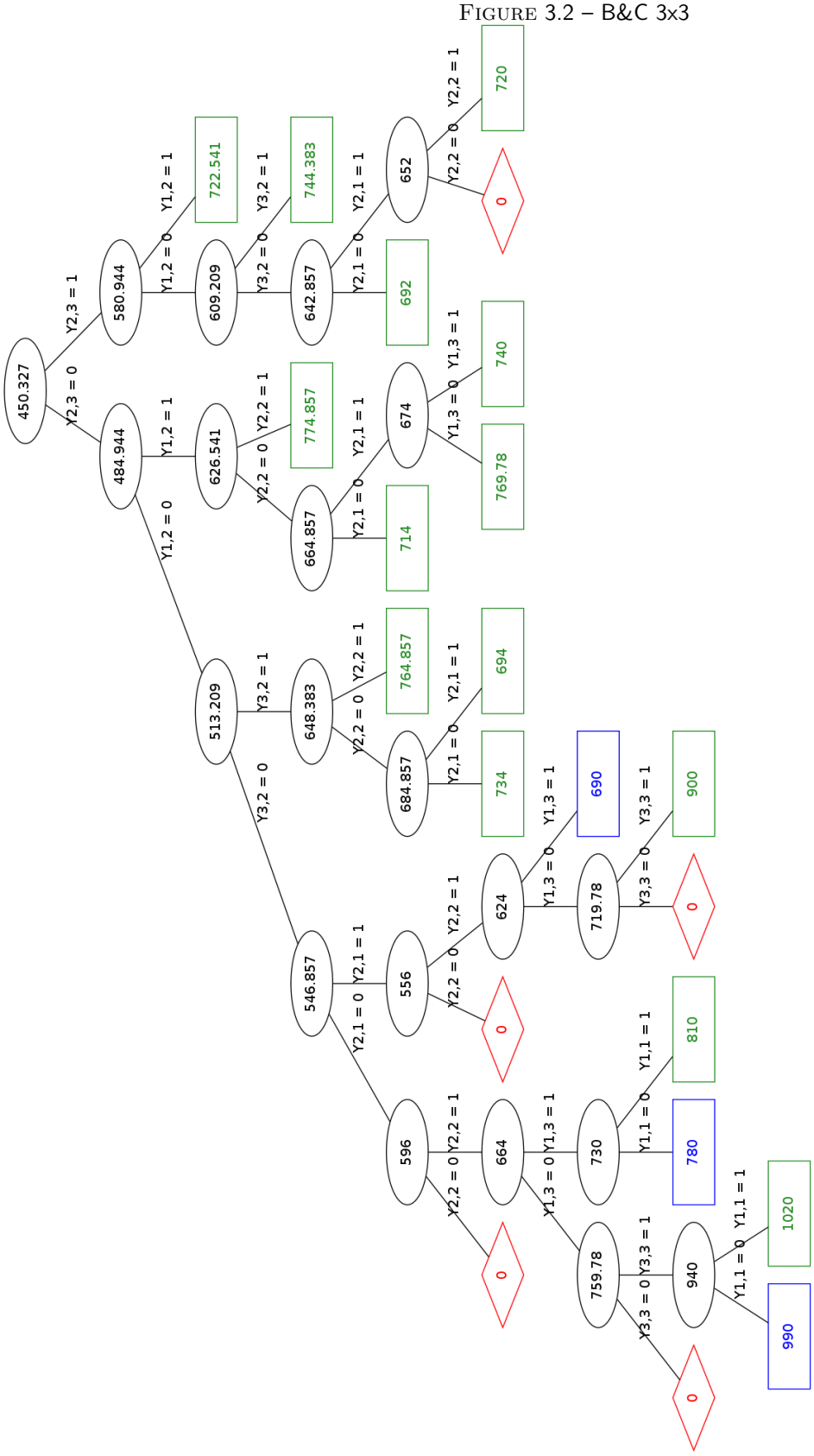
$$
\begin{array}{lll}
x_1^1 = 0 & x_1^2 = 0 & x_1^3 = 90 \\
x_2^1 = 120 & x_2^2 = 110 & x_2^3 = 0 \\
x_3^1 = 0 & x_3^2 = 0 & x_3^3 = 0
\end{array}
$$

$$
\begin{array}{lll}
s_1^1 = 0 & s_1^2 = 0 & s_1^3 = 50 \\
s_2^1 = 50 & s_2^2 = 50 & s_2^3 = 0 \\
s_3^1 = 0 & s_3^2 = 0 & s_3^3 = 0
\end{array}
$$

$$
\begin{array}{lll}
r_1^1 = 0 & r_1^2 = 20 & r_1^3 = 0 \\
r_2^1 = 0 & r_2^2 = 0 & r_2^3 = 0 \\
r_3^1 = 0 & r_3^2 = 0 & r_3^3 = 0
\end{array}
$$

$$
\begin{array}{lll}
y_1^1 = 0 & y_1^2 = 0 & y_1^3 = 1 \\
y_2^1 = 1 & y_2^2 = 1 & y_2^3 = 0 \\
y_3^1 = 0 & y_3^2 = 0 & y_3^3 = 0
\end{array}
$$

FIGURE 3.2 – B&C 3x3



3.2.3 Taille 4 x 4

Taille maximale de l'arbre : 131 071 nœuds. Trouvé $z = 1660$ en 1617 itérations.

$$
\begin{array}{llll}
x_1^1 = 0 & x_1^2 = 0 & x_1^3 = 90 & x_1^4 = 0 \\
x_2^1 = 120 & x_2^2 = 120 & x_2^3 = 0 & x_2^4 = 0 \\
x_3^1 = 0 & x_3^2 = 0 & x_3^3 = 0 & x_3^4 = 350 \\
x_4^1 = 100 & x_4^2 = 0 & x_4^3 = 100 & x_4^4 = 0
\end{array}
$$

$$
\begin{array}{llll}
s_1^1 = 0 & s_1^2 = 0 & s_1^3 = 50 & s_1^4 = 0 \\
s_2^1 = 50 & s_2^2 = 60 & s_2^3 = 0 & s_2^4 = 0 \\
s_3^1 = 0 & s_3^2 = 10 & s_3^3 = 0 & s_3^4 = 150 \\
s_4^1 = 0 & s_4^2 = 0 & s_4^3 = 0 & s_4^4 = 0
\end{array}
$$

$$
\begin{array}{llll}
r_1^1 = 0 & r_1^2 = 20 & r_1^3 = 0 & r_1^4 = 0 \\
r_2^1 = 0 & r_2^2 = 0 & r_2^3 = 0 & r_2^4 = 100 \\
r_3^1 = 0 & r_3^2 = 0 & r_3^3 = 0 & r_3^4 = 0 \\
r_4^1 = 0 & r_4^2 = 0 & r_4^3 = 0 & r_4^4 = 0
\end{array}
$$

$$
\begin{array}{llll}
y_1^1 = 0 & y_1^2 = 0 & y_1^3 = 1 & y_1^4 = 0 \\
y_2^1 = 1 & y_2^2 = 1 & y_2^3 = 0 & y_2^4 = 0 \\
y_3^1 = 0 & y_3^2 = 0 & y_3^3 = 0 & y_3^4 = 1 \\
y_4^1 = 1 & y_4^2 = 0 & y_4^3 = 1 & y_4^4 = 0
\end{array}
$$

La taille de l'arbre est trop importante pour l'afficher.

3.2.4 Taille 5 x 5

Taille maximale de l'arbre : 67 108 863 nœuds. Dépassement mémoire.