VIỆN QUỐC TẾ PHÁP NGỮ
Institut Francophone International

RAPPORT DE TP3 PROGRAMMATION PAR CONTRAINTE

Résoudre le Problème de BINPA-CKING2D avec OpenCBLS et Choco

Etudiant :

KAFANDO Rodrique

Professeur:

Dung Pham Quang

Promotion 21

INTRODUCTION

Dans ce TP3, il est question pour nous de résoudre un problème de BinPacking2D en utilisant OpenCBLS et Choco. En effet, le problème du binpacking2D consiste à classer des objets dans une surface sans qu'il n'y ait débordement ou de superposition entre objets. Pour ce faire, nous présenterons tout d'abord la problématique du sujet, ensuite nous définirons les variables et leur domaine de définition, les contraintes qui existent entre les variables. En outre nous présenterons les résultats issus de l'implémentation des deux librairies et nous terminerons par une étude comparative des résultats obtenus.

I) PROBLÉMATIQUE

Comme énoncé dans le sujet, le problème, il s'agit d'un conteneur ayant une largeur W et une hauteur H. Il existe N rectangles de 1, 2,..., N ou le i-ème rectangle possède une largeur **wi** et une hauteur **hi**.
Il est donc question de trouver une combinaison possible qui permettra de placer ces objets dans le conteneur tel que :

- Les cotés des objets soient disposés parallèlement aux cotes du conteneur ;
- Les objets peuvent être pivotes de 90, 180 ou 270 degré ;
- Deux objets ne doivent pas se superposer

Les entrées sont constituées comme suit :

- La première ligne contient deux entiers W et H, respectivement la largeur et la hauteur

du conteneur,

- Chaque ligne suivante de i+1, terminée par -1, contient deux entiers w i et h i

respectivement la largeur et la hauteur de l'objet i.
Afin de pouvoir visualiser les résultats obtenus, la sortie de la résolution serra présentée sous forme HTML comme l'exemple ci-dessous.

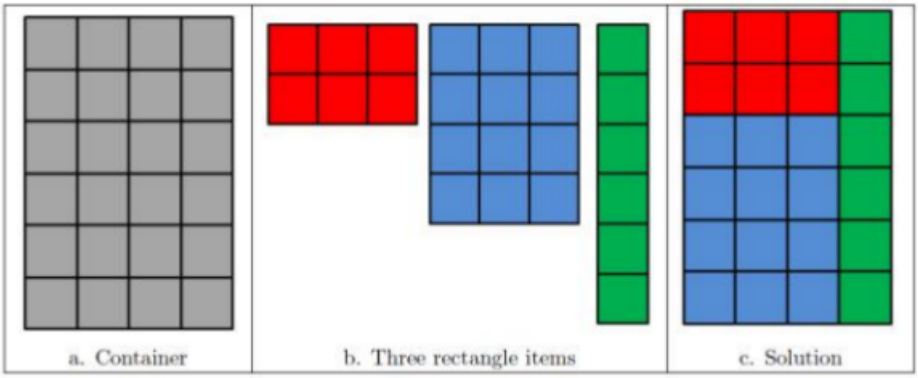

Figure1 : Exemple de résolution

II) Modèle mathématique du problème

II-1) Les variables

Soit i ∈ 1, ..., N, le i-ème rectangle à placer dans le conteneur :

- Chaque rectangle dans le conteneur est définie par les coordonnées **(Xi , Yi)** ;
- La variable de rotation **Oi**, (Oi=0 s'il n'y a pas d'orientation et Oi=1 s'il y a orientation).

II-2) les contraintes

Deux principales contraintes sont à prendre en compte.

- Deux rectangles ne doivent jamais se superposer quelque soit l'orientation

Pour ce faire, nous définissons les quatre cas de figure que nous pouvons faire face.
Soit ∀i, j ∈ {1, ..., N } on a:

1

$oi = 0 \text{ et } oj = 0 \Rightarrow$

$x_i + w_i \le x_j$ ou $x_j + w_j \le x_i$ ou $y_j + h_j \le y_i$ ou $y_i + h_i \le y_j$

3

$oi = 1 \text{ et } oj = 0 \Rightarrow$

$x_i + h_i \le x_j$ ou $x_j + w_j \le x_i$ ou $y_j + h_j \le y_i$ ou $y_i + w_i \le y_j$

2

$oi = 0 \text{ et } oj = 1 \Rightarrow$

$x_i + w_i \le x_j$ ou $x_j + h_j \le x_i$ ou $y_j + h_j \le y_i$ ou $y_i + h_i \le y_j$

4

$oi = 1 \text{ et } oj = 1 \Rightarrow$

$x_i + h_i \le x_j$ ou $x_j + h_j \le x_i$ ou $y_j + w_j \le y_i$ ou $y_i + w_i \le y_j$

- Aucun rectangle ne doit déborder le conteneur

Dans ces circonstances, nous distinguons principalement deux (02) cas possible:
Soit ∀i ∈ {1, ..., N } on a :

1

$Oi = 0 \Rightarrow$

$x_i + w_i \le W$ et $y_i + h_i \le H$

2

$Oi = 1 \Rightarrow$

$x_i + h_i \le W$ et $y_i + w_i \le H$

II-3) Domaine des variables

Les variables Xi, Yi et Oi sont définies dans les domaines suivantes :
Soit ∀i ∈ {1, ..., N } on a :

- Domaine(x i) = {0, ..., W } ;
- Domaine(x i) = {0, ..., H} ;
- Domaine(o i) = {0, 1}.

III-) IMPLÉMENTATION

III-1) Implémentation avec Choco

- Déclaration des variables

```
//déclaration des variables
IntegerVariable[] x = new IntegerVariable[n];
IntegerVariable[] y = new IntegerVariable[n];
IntegerVariable[] o = new IntegerVariable[n];

x[i] = Choco.makeIntVar("x[" + i + "]", 0, W);
y[i] = Choco.makeIntVar("y[" + i + "]", 0, H);
o[i] = Choco.makeIntVar("o[" + i + "]", 0, 1);
```

- Contraintes de non-superposition

```
// Contraintes de non superposition
// o[i] = 0, o[j] = 0
m.addConstraint(Choco.implies(
        Choco.and(Choco.eq(o[i], 0), Choco.eq(o[j], 0)),
        Choco.or(Choco.leq(Choco.plus(x[i], w[i]), x[j]),
        Choco.leq(Choco.plus(x[j], w[j]), x[i]),
        Choco.leq(Choco.plus(y[j], h[j]), y[i]),
        Choco.leq(Choco.plus(y[i], h[i]), y[j]))));
// o[i] = 0, o[j] = 1
m.addConstraint(Choco.implies(
        Choco.and(Choco.eq(o[i], 0), Choco.eq(o[j], 1)),
        Choco.or(Choco.leq(Choco.plus(x[i], w[i]), x[j]),
        Choco.leq(Choco.plus(x[j], h[j]), x[i]),
        Choco.leq(Choco.plus(y[j], w[j]), y[i]),
        Choco.leq(Choco.plus(y[i], h[i]), y[j]))));

// o[i] = 1, o[j] = 0
m.addConstraint(Choco.implies(
        Choco.and(Choco.eq(o[i], 1), Choco.eq(o[j], 0)),
        Choco.or(Choco.leq(Choco.plus(x[i], h[i]), x[j]),
        Choco.leq(Choco.plus(x[j], w[j]), x[i]),
        Choco.leq(Choco.plus(y[j], h[j]), y[i],
        Choco.leq(Choco.plus(y[i], w[i]), y[j]))));
```

```
// o[i] = 1, o[j] = 1
m.addConstraint(Choco.implies(
        Choco.and(Choco.eq(o[i], 1), Choco.eq(o[j], 1)),
        Choco.or(Choco.leq(Choco.plus(x[i], h[i]), x[j]),
        Choco.leq(Choco.plus(x[j], h[j]), x[i]),
        Choco.leq(Choco.plus(y[j], w[j]), y[i]),
        Choco.leq(Choco.plus(y[i], w[i]), y[j]))));
```

- Contraintes de non-dépassement du conteneur

```
// constraintes de non depassement du conteneur
// o[i] = 0
m.addConstraint(Choco.implies(Choco.eq(o[i], 0),
        Choco.and(Choco.leq(Choco.plus(x[i], w[i]), W),
        Choco.leq(Choco.plus(y[i], h[i]), H))));
// o[i] = 1
m.addConstraint(Choco.implies(Choco.eq(o[i], 1),
        Choco.and(Choco.leq(Choco.plus(x[i], h[i]), W),
        Choco.leq(Choco.plus(y[i], w[i]), H))));
```

III-2) Implémentation avec OpenCBLS

- Déclaration des variables

```
localsrch = new LocalSearchManager();
S = new ConstraintSystem(localsrch);
y = new VarIntLS[n];
x = new VarIntLS[n];
o = new VarIntLS[n];

x[i] = new VarIntLS(localsrch, 0, W);
o[i] = new VarIntLS(localsrch, 0, 1);
y[i] = new VarIntLS(localsrch, 0, H);
```

- Contraintes de non-superposition

```
// o[i] = 0, o[j] = 0
IConstraint[] contrainte1 = new IConstraint[4];
contrainte1[0] = new LessOrEqual(new FuncPlus(x[i], w[i]), x[j]);
contrainte1[1] = new LessOrEqual(new FuncPlus(x[j], w[j]), x[i]);
contrainte1[2] = new LessOrEqual(new FuncPlus(y[j], h[j]), y[i]);
contrainte1[3] = new LessOrEqual(new FuncPlus(y[i], h[i]), y[j]);
S.post(new Implicate(new AND(new IsEqual(o[i], 0),
                        new IsEqual(o[j], 0)), new OR(contrainte1)));
// o[i] = 1, o[j] = 0
IConstraint[] contrainte2 = new IConstraint[4];
contrainte2[0] = new LessOrEqual(new FuncPlus(x[i], h[i]), x[j]);
contrainte2[1] = new LessOrEqual(new FuncPlus(x[j], w[j]), x[i]);
contrainte2[2] = new LessOrEqual(new FuncPlus(y[j], h[j]), y[i]);
contrainte2[3] = new LessOrEqual(new FuncPlus(y[i], w[i]), y[j]);
S.post(new Implicate(new AND(new IsEqual(o[i], 1),
                        new IsEqual(o[j], 0)), new OR(contrainte2)));
```

```
// o[i] = 0, o[j] = 1
IConstraint[] contrainte3 = new IConstraint[4];
contrainte3[0] = new LessOrEqual(new FuncPlus(x[i], w[i]), x[j]);
contrainte3[1] = new LessOrEqual(new FuncPlus(x[j], h[j]), x[i]);
contrainte3[2] = new LessOrEqual(new FuncPlus(y[j], w[j]), y[i]);
contrainte3[3] = new LessOrEqual(new FuncPlus(y[i], h[i]), y[j]);
S.post(new Implicate(new AND(new IsEqual(o[i], 0),
                        new IsEqual(o[j], 1)), new OR(contrainte3)));

// o[i] = 1, o[j] = 1
IConstraint[] contrainte4 = new IConstraint[4];
contrainte4[0] = new LessOrEqual(new FuncPlus(x[i], h[i]), x[j]);
contrainte4[1] = new LessOrEqual(new FuncPlus(x[j], h[j]), x[i]);
contrainte4[2] = new LessOrEqual(new FuncPlus(y[j], w[j]), y[i]);
contrainte4[3] = new LessOrEqual(new FuncPlus(y[i], w[i]), y[j]);
S.post(new Implicate(new AND(new IsEqual(o[i], 1),
                        new IsEqual(o[j], 1)), new OR(contrainte4)));
```

- Contraintes de non-dépassement du conteneur

```
// constraintes de non-dépassement du conteneur
// o[i] = 0
IConstraint[] contraintOo = new IConstraint[2];
contraintOo[0] = new LessOrEqual(new FuncPlus(x[i], w[i]), W);
contraintOo[1] = new LessOrEqual(new FuncPlus(y[i], h[i]), H);
S.post(new Implicate(new IsEqual(o[i], 0), new AND(contraintOo)));
// o[i] = 1
IConstraint[] contraintO1 = new IConstraint[2];
contraintO1[0] = new LessOrEqual(new FuncPlus(x[i], h[i]), W);
contraintO1[1] = new LessOrEqual(new FuncPlus(y[i], w[i]), H);
S.post(new Implicate(new IsEqual(o[i], 1), new AND(contraintO1)));
```

IV-) Tableau présentant les résultats expérimentaux

Dans le but de pouvoir évaluer la pertinence de notre programme, nous avons effectué des tests sur des données différente et les résultats obtenus sont les suivants.
Nous les illustrons dans le tableau comparative à l'issu du quel nous précisons par **oui** si une solution a été trouvé et par **non** si non.

DataFiles	Choco_Results		OpenCBLS_Results	
bin-packing-2D-W10-H7-I6.txt	825 ms	True	372 ms	True
bin-packing-2D-W10-H8-I6.txt	587 ms	True	130 ms	True
bin-packing-2D-W19-H16-I21.txt	--	False	3000194 ms	False
bin-packing-2D-W19-H17-I21.txt	28372 ms	True	1638.0 ms	True
bin-packing-2D-W19-H18-I21.txt	22652 ms	True	901 ms	True
bin-packing-2D-W19-H19-I21.txt	108363 ms	True	536 ms	True
bin-packing-2D-W19-H20-I21.txt	16380 ms	True	540 ms	True
bin-packing-2D-W19-H21-I21.txt	36487 ms	True	624 ms	True
bin-packing-2D-W19-H22-I21.txt	83751 ms	True	640 ms	True
bin-packing-2D-W19-H23-I21.txt	185773 ms	True	552 ms	True
bin-packing-2D-W19-H24-I21.txt	224665 ms	True	362 ms	True

CONCLUSION

La programmation par contraintes est un formidable paradigme déclaratif permettant la modélisation et la résolution de nombreux problèmes de complexités diverses.
Dans ce travail nous avons évalué d'une part les performance de la librairie choco et OpenCBLS à partir du problème de binpacking2D. A partir des résultats obtenus, nous remarquons que la librairie OpenCBLS offre un temps d'exécution plus court que Choco avec les mêmes données en entrée même si certaines données n'ont pas pu être traités. En revanche, les faux résultats que nous avons obtenu ne signifie forcement pas qu'il n'y a pas de solution, mais que le problème n'a pas pu être résolu dans le temps limite.