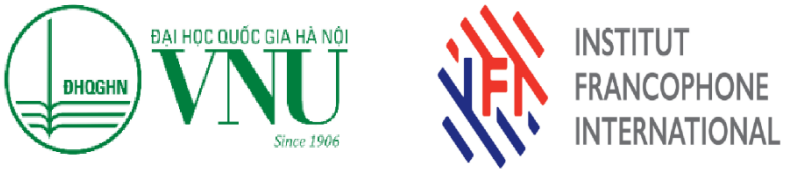
Institut Francophone International
(IFI)



Cours de Programmation par Contrainte

Professeur : Dung Pham Quang

TP3: Bin Packing

Préparé par :

ELIODOR Ednalson Guy Mirlin – P21

Date : 19 Septembre 2017

1- Introduction

De manière formelle, le problème de bin-packing à deux (2) dimensions est défini comme suivant :

Etant donné un ensemble de n objets rectangulaires A= {a1,..., an} et un nombre illimité de rectangles identiques (les bins) de dimensions plus larges que celles des objets, le problème consiste à déterminer le nombre minimum de bins à utiliser pour ranger l'ensemble des objets sans chevauchement.

Le problème de Bin Packing à deux dimensions est une généralisation du problème de bin packing unidimensionnel 1BP. La définition du problème peut être généralisée à N dimensions avec des objets eux à N dimensions.

Les problèmes de Bin packing possèdent de très grand nombre d'application dans les domaines informatiques et industriels telles que le remplissage de véhicules avec contrainte de poids, le placement dans un entrepôt, le rangement d'objet dans des boites, véhicules, dans l'industrie de l'acier, du bois etc.

a. Spécification du problème

Dans le cadre de ce TP, nous avons résolu le problème de BIN PACKING 2D avec utilisation de la programmation dynamique. Nous avons dans ce cas utilisé deux librairies java: OpenCBLS et Choco pour l'implémentation. Le modèle du problème sera donné avec une formulation mathématique. Les programmes seront fournis en annexe. Une étude comparative entre les deux librairies sera ensuite faite avec le temps de recherche comme paramètre.

Formulation du problème :

Le conteneur est de largeur W et de hauteur H. Il y a N rectangles de {1,2,...,N}. Chaque rectangle i a une largeur wi et une hauteur hi. L'objectif est de trouver un moyen de placer les N rectangles dans le conteneur ainsi : Les côtés des rectangles doivent être parallèles avec les côtés du conteneur, aucun rectangle ne doit chevaucher l'autre dans le conteneur, il peut y avoir rotations de 90 degrés qui n'est pas forcément obligatoire.

2- Spécification des variables et Contraintes

Comme on a donc précisé dans le point « Formulation du problème », les rectangles à placer dans le conteneur partent de 1 à N, ainsi pour i allant de 1 à N, (xi, yi) représente donc les coordonnées du rectangle i dans le conteneur. On a aussi Oi lorsque i=1 (Orientation) et i=0 (pas d'orientation).

Maintenant parlons des contraintes :

a. Contrainte de non-superposition

Dans ce type de contrainte, il faut préciser que 2 rectangles présents dans le conteneur ne doivent jamais être superposés. Ainsi, 4 cas peuvent se présenter :

Cas 1 :

$$o_i = 0 \text{ et } o_j = 0 \Rightarrow \begin{cases} x_i + w_i \leq x_j \\ \text{ou} \\ x_j + w_j \leq x_i \\ \text{ou} \\ y_j + h_j \leq y_i \\ \text{ou} \\ y_i + h_i \leq y_j \end{cases}$$

Cas 2 :

$$o_i = 0 \text{ et } o_j = 1 \Rightarrow \begin{cases} x_i + w_i \leq x_j \\ \text{ou} \\ x_j + h_j \leq x_i \\ \text{ou} \\ y_j + h_j \leq y_i \\ \text{ou} \\ y_i + h_i \leq y_j \end{cases}$$

Cas 3 :

$$o_i = 1 \text{ et } o_j = 0 \Rightarrow \begin{cases} x_i + h_i \leq x_j \\ \text{ou} \\ x_j + w_j \leq x_i \\ \text{ou} \\ y_j + h_j \leq y_i \\ \text{ou} \\ y_i + w_i \leq y_j \end{cases}$$

Cas 4 :

$$o_i = 1 \text{ et } o_j = 1 \Rightarrow \begin{cases} x_i + h_i \leq x_j \\ \text{ou} \\ x_j + h_j \leq x_i \\ \text{ou} \\ y_j + w_j \leq y_i \\ \text{ou} \\ y_i + w_i \leq y_j \end{cases}$$

b. Les rectangles doivent se placer forcément à l'intérieur des conteneurs

Cette contrainte permet aux différents triangles de se positionner dans la grille (Conteneur). On distingue ainsi 2 cas pour i allant de 1 à N, avec xi allant de 0 a W, y_i de 0 à H, et O_i pour i allant de 0 à 1 :

Cas 1 :

$$o_i = 0 \Rightarrow \begin{cases} x_i + w_i \leq W \\ \text{et} \\ y_i + h_i \leq H \end{cases}$$

Cas 2 :

$$o_i = 1 \Rightarrow \begin{cases} x_i + h_i \leq W \\ \text{et} \\ y_i + w_i \leq H \end{cases}$$

Implémentation avec CHOCO :

```
import java.util.*;
import java.io.*;

import choco.cp.model.*;
import choco.cp.solver.CPSolver;
import choco.kernel.model.variables.integer.*;
import choco.Choco;
```

```
public class Binpacking2D {

    public int W, H;
    public int n;
    public int[] w;
    public int[] h;
    public int[] s_x;
    public int[] s_y;
    public int[] s_o;

    public Binpacking2D() {

    }

    public void readData(String fn) {
        try {
            Scanner in = new Scanner(new File(fn));
            W = in.nextInt();
            H = in.nextInt();
            ArrayList<Item> I = new ArrayList<Item>();
            while (true) {
                int wi = in.nextInt();
                if (wi == -1)
                    break;
                int hi = in.nextInt();
                I.add(new Item(wi, hi));
            }
            in.close();

            n = I.size();
            w = new int[n];
            h = new int[n];
            for (int i = 0; i < I.size(); i++) {
                w[i] = I.get(i).w;
                h[i] = I.get(i).h;
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
public boolean solveWithOrientation(int timeLimit) {

    timeLimit = timeLimit * 1000;

    CPModel m = new CPModel();
    IntegerVariable[] x = new IntegerVariable[n];
    IntegerVariable[] y = new IntegerVariable[n];
    IntegerVariable[] o = new IntegerVariable[n];
    for (int i = 0; i < n; i++) {
        x[i] = Choco.makeIntVar("x[" + i + "]", 0, W);
        y[i] = Choco.makeIntVar("y[" + i + "]", 0, H);
        o[i] = Choco.makeIntVar("o[" + i + "]", 0, 1);
    }
    for (int i = 0; i < n; i++) {
        // constraint on inside the grid
        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))));

        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))));
    }
```

```
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            // o[i] = 0, o[j] = 0
            // constraint on overlap
            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
            // constraint on overlap
            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
            // constraint on overlap
            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]))));

        }
    }
```

```
    CPSolver s = new CPSolver();
    s.read(m);
    s.setTimeLimit(timeLimit);
    boolean ok = false;
    // ChocoLogging.toSolution();
    ok = s.solve();

    System.out.println("solution    =  " + ok);
    // si la solution est bonne
    if (ok) {
        s_x = new int[n];
        s_y = new int[n];
        s_o = new int[n];

        for (int i = 0; i < n; i++) {
            s_x[i] = s.getVar(x[i]).getVal();
            s_y[i] = s.getVar(y[i]).getVal();
            s_o[i] = s.getVar(o[i]).getVal();
        }

        // System.out.println("  " + s.toString());

        for (int i = 0; i < n; i++) {
            System.out.println("Item " + i + " at ("
                    + s.getVar(x[i]).getVal() + ","
                    + s.getVar(y[i]).getVal() + ")" + "  "
                    + s.getVar(o[i]).getVal());
        }
    }
    return ok;
}
```

Implémentation avec CBLS

```
import localsearch.model.IConstraint;
import localsearch.model.LocalSearchManager;
import localsearch.model.VarIntLS;
import localsearch.search.TabuSearch;
import localsearch.selectors.MinMaxSelector;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Scanner;

import choco.Choco;

class Item {
    int w, h;

    public Item(int w, int h) {
        this.w = w;
        this.h = h;
    }
}

public class BinpackingCBLS2D {
    public int W, H;
    public int n;
    public int[] w;
    public int[] h;

    LocalSearchManager ls;
    ConstraintSystem S;
    VarIntLS[] x;
    VarIntLS[] y;
    VarIntLS[] o;

    public BinpackingCBLS2D() {
    }

    public void readData(String fn) {
        try {
            Scanner in = new Scanner(new File(fn));
            W = in.nextInt();
            H = in.nextInt();
            ArrayList<Item> I = new ArrayList<Item>();
            while (true) {
                int wi = in.nextInt();
                if (wi == -1)
                    break;
                int hi = in.nextInt();
                I.add(new Item(wi, hi));
            }
            in.close();

            n = I.size();
            w = new int[n];
            h = new int[n];
            for (int i = 0; i < I.size(); i++) {
                w[i] = I.get(i).w;
                h[i] = I.get(i).h;
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```

```
public void stateModel() {
    ls = new LocalSearchManager();
    S = new ConstraintSystem(ls);
    y = new VarIntLS[n];
    x = new VarIntLS[n];
    o = new VarIntLS[n];
    for (int i = 0; i < n; i++) {
        x[i] = new VarIntLS(ls, 0, W);
        o[i] = new VarIntLS(ls, 0, 1);
        y[i] = new VarIntLS(ls, 0, H);
    }
    for (int i = 0; i < n; i++) {
        // constraint on inside the grid
        IConstraint[] c5 = new IConstraint[2];
        c5[0] = new LessOrEqual(new FuncPlus(x[i], w[i]), W);
        c5[1] = new LessOrEqual(new FuncPlus(y[i], h[i]), H);
        S.post(new Implicate(new IsEqual(o[i], 0), new AND(c5)));

        IConstraint[] c6 = new IConstraint[2];
        c6[0] = new LessOrEqual(new FuncPlus(x[i], h[i]), W);
        c6[1] = new LessOrEqual(new FuncPlus(y[i], w[i]), H);
        S.post(new Implicate(new IsEqual(o[i], 1), new AND(c6)));
    }
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {

            // o[i] = 0, o[j] = 0
            IConstraint[] c1 = new IConstraint[4];
            c1[0] = new LessOrEqual(new FuncPlus(x[i], w[i]), x[j]);
            c1[1] = new LessOrEqual(new FuncPlus(x[j], w[j]), x[i]);
            c1[2] = new LessOrEqual(new FuncPlus(y[j], h[j]), y[i]);
            c1[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(c1)));
            // o[i] = 1, o[j] = 0
            IConstraint[] c2 = new IConstraint[4];
            c2[0] = new LessOrEqual(new FuncPlus(x[i], h[i]), x[j]);
            c2[1] = new LessOrEqual(new FuncPlus(x[j], w[j]), x[i]);
            c2[2] = new LessOrEqual(new FuncPlus(y[j], h[j]), y[i]);
            c2[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(c2)));

            // o[i] = 0, o[j] = 1
            IConstraint[] c3 = new IConstraint[4];
            c3[0] = new LessOrEqual(new FuncPlus(x[i], w[i]), x[j]);
            c3[1] = new LessOrEqual(new FuncPlus(x[j], h[j]), x[i]);
            c3[2] = new LessOrEqual(new FuncPlus(y[j], w[j]), y[i]);
            c3[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(c3)));

            // o[i] = 1, o[j] = 1
            IConstraint[] c4 = new IConstraint[4];
            c4[0] = new LessOrEqual(new FuncPlus(x[i], h[i]), x[j]);
            c4[1] = new LessOrEqual(new FuncPlus(x[j], h[j]), x[i]);
            c4[2] = new LessOrEqual(new FuncPlus(y[j], w[j]), y[i]);
            c4[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(c4)));
        }
    }
    ls.close();
}
```

```java
public void search(int timeLimit) {
    long timelimit = timeLimit * 1000;
    long timeInit = System.currentTimeMillis();
    MinMaxSelector mms = new MinMaxSelector(S);
    // int it = 0;
    while (System.currentTimeMillis() - timeInit < timelimit
            && S.violations() > 0) {
        VarIntLS sel_x = mms.selectMostViolatingVariable();
        int sel_v = mms.selectMostPromissingValue(sel_x);
        sel_x.setValuePropagate(sel_v);
        // it++;
        // System.out.println("étape    " + it + "   S = " +
        // S.violations());
    }
```

Etudes comparatives entre les deux méthodes :

Il faut préciser que ces programmes ont été testés sur notre machine ayant les configurations suivantes : Intel Core i3 avec 1.2 GHz. Les tests ont été effectués à partir des fichiers de données en entrée des différents programmes, on a essayé aussi de changer le temps afin de voir les variations des résultats.

Ainsi pour t= 300 Secondes on a trouvé :

Fichier en entrée	CBLS	CHOCO
bin-packing-2DW10-H7-I6	Oui	Oui
bin-packing-2DW10-H8-I6	Oui	Oui
bin-packing-2DW19-H16-I21	Non	Non
bin-packing-2DW19-H17-I21	Non	Oui
bin-packing-2DW19-H18-I21	Oui	Oui
bin-packing-2DW19-H19-I21	Oui	Oui
bin-packing-2DW19-H20-I21	Oui	Oui
bin-packing-2DW19-H21-I21	Oui	Oui
bin-packing-2DW19-H22-I21	Oui	Oui
bin-packing-2DW19-H23-I21	Oui	Oui
bin-packing-2DW19-H24-I21	Oui	Oui

Avec t = 100 Secondes

Fichier en entrée	CBLS	CHOCO
bin-packing-2DW10-H7-I6	Oui	Oui
bin-packing-2DW10-H8-I6	Oui	Oui
bin-packing-2DW19-H16-I21	Non	Non
bin-packing-2DW19-H17-I21	Non	Oui
bin-packing-2DW19-H18-I21	Non	Oui
bin-packing-2DW19-H19-I21	Non	Non
bin-packing-2DW19-H20-I21	Non	Oui
bin-packing-2DW19-H21-I21	Oui	Oui
bin-packing-2DW19-H22-I21	Oui	Non
bin-packing-2DW19-H23-I21	Oui	Non
bin-packing-2DW19-H24-I21	Oui	Non

Pour t= 10 Secondes

Fichier en entrée	CBLS	CHOCO
bin-packing-2DW10-H7-I6	Oui	Oui
bin-packing-2DW10-H8-I6	Oui	Oui
bin-packing-2DW19-H16-I21	Non	Non
bin-packing-2DW19-H17-I21	Non	Non
bin-packing-2DW19-H18-I21	Non	Non
bin-packing-2DW19-H19-I21	Non	Non
bin-packing-2DW19-H20-I21	Non	Non
bin-packing-2DW19-H21-I21	Oui	Non
bin-packing-2DW19-H22-I21	Oui	Non
bin-packing-2DW19-H23-I21	Oui	Non
bin-packing-2DW19-H24-I21	Oui	Non

Conclusion

Nous avons conçu un modèle permettant de résoudre ce problème tout en respectant les contraintes fixées. Les résultats qu'on a obtenu montrent qu'avec CHOCO on a eu 18/33 (Oui) tandis qu'avec CBLS on a eu 21/33 (Oui). On peut conclure que CBLS est meilleurs que CHOCO dans la résolution de ce problème.