Etude de cas : Résolution du problème d'ordonnancement de véhicules en Choco

Réalisé par le binôme :

ZARROUQ Soukaina

BANAH Fathiya

Filière :

DSE

L'Objectif

Utiliser Choco Solver pour la modélisation et la résolution du problème d'ordonnancement de véhicules.

Après avoir créé une classe de "CSVPOV", nous avons défini et initialisé les variables suivantes à l'aide du :

Définition

Soit le quintuplait (V, O, p, q, r) avec :
V : Ensemble des voitures à produire.
O : Ensemble des options disponibles.
p, q : Listes d'entier, pour chaque option i, p_i/q_i représente la fréquence d'apparition.
r : Matrice de booléen $r_{i,j} = 1$, représente le fait que l'option O_j est présente sur le véhicule V_i, 0 sinon.

Nous avons travaillé avec les valeurs suivantes :

		Catégories											
Option	contrainte	1	2	3	4	5	6	7	8	9	10	11	12
1	1/2		•	•				•	•	•			•
2	2/3	•		•	•	•		•				•	•
3	1/3		•					•		•	•	•	
4	2/5				•		•		•				•
5	1/5		•			•							
	nb de voitures	3	1	2	4	3	3	2	1	1	2	2	1

module-info.java | CSVPOV.java

```java
import choco.Choco;

public class CSVPOV {

    public CSVPOV() {
        this.init();
        this.NbreVConstraint();
        this.OptionConstraint();
        this.SeqConstraint();
        this.solve();
    }

    int v = 25; // Nombre de voitures à produire
    int o = 5; // Nombre des options disponibles
    int c = 12; // Nombre de catégories

    // Listes d'entier pour chaque option i; p[i]/q[i] = la fréquence d'apparition
    int[] p = {1,2,1,2,1};
    int[] q = {2,3,3,5,5};

    int[] d = {3,1,2,4,3,3,2,1,1,2,2,1}; // demande
    int[][] r = {
            {0,1,0,0,0},
            {1,0,1,0,1},
            {1,1,0,0,0},
            {0,1,0,1,0},
            {0,1,0,0,1},
            {0,0,0,1,0},
            {1,1,1,0,0},
            {1,0,0,1,0},
            {1,0,1,0,0},
            {0,0,1,0,0},
            {0,1,1,0,0},
            {1,1,0,1,0}
    }; //  Matrice de booléen ri,j = 1, représente le fait que l'option Oj est présente sur le véhicule Vi, 0 sinon.

```

Le Model, accompagné du Solver, est l'un des éléments fondamentaux de choco. Il permet de décrire simplement un problème de manière déclarative. Son rôle principal est d'enregistrer les variables et contraintes d'un modèle.

```java
    Model model = new CPModel();
    Solver solver = new CPSolver();

```

Nous avons ajouté 2 variables et leurs domaines :

```java

    private IntegerVariable[]  categoV;
    private IntegerVariable[][] optionV;

    public void init() {
        this.categoV = makeIntVarArray("classement Voiture", this.v, 1, this.c);
        this.optionV =  makeIntVarArray("optionV", this.o, this.v, 0, 1);
    }

```

La variable categoV : un tableau qui présente le type de catégorie de chaque voiture.

La variable optionV : un tableau à deux dimensions qui indique si la voiture possède une certaine option ou pas (1 ou 0).

➢ Les contraintes :

L'ajout d'une contrainte (addConstraint) au modèle ajoute automatiquement toutes ses variables, c'est-à-dire il n'est pas nécessaire de les ajouter explicitement

À cet ensemble, s'ajoute les contraintes de capacité du problème ainsi que la demande pour chacune des catégories de véhicule :

NbreVConstraint()

Cette contrainte nécessite que le nombre de voitures dans chaque catégorie et le nombre de voiture dans que nous avons défini précédemment doivent être égaux

```
public void NbreVConstraint() {
    for(int i = 0; i < c; i++) {
        this.model.addConstraint(Choco.occurrence(d[i], this.categoV, i+1));
    }
}
```

OptionConstraint()

Cette contrainte implique que le nombre d'option de chaque voiture et le nombre de catégorie doivent être égaux

```
public void OptionConstraint() {
    for (int i = 0; i < this.c; i++) {

        for (int j = 0; j < this.v; j++) {
            Constraint[] C = new Constraint[this.o];
            for (int k = 0; k < this.o; k++) {
                C[k] = eq(this.optionV[k][j], r[i][k]);
            }

            model.addConstraint(implies(eq(this.categoV[j], i+1),and(C)));
        }
    }
}
```

SeqConstraint()

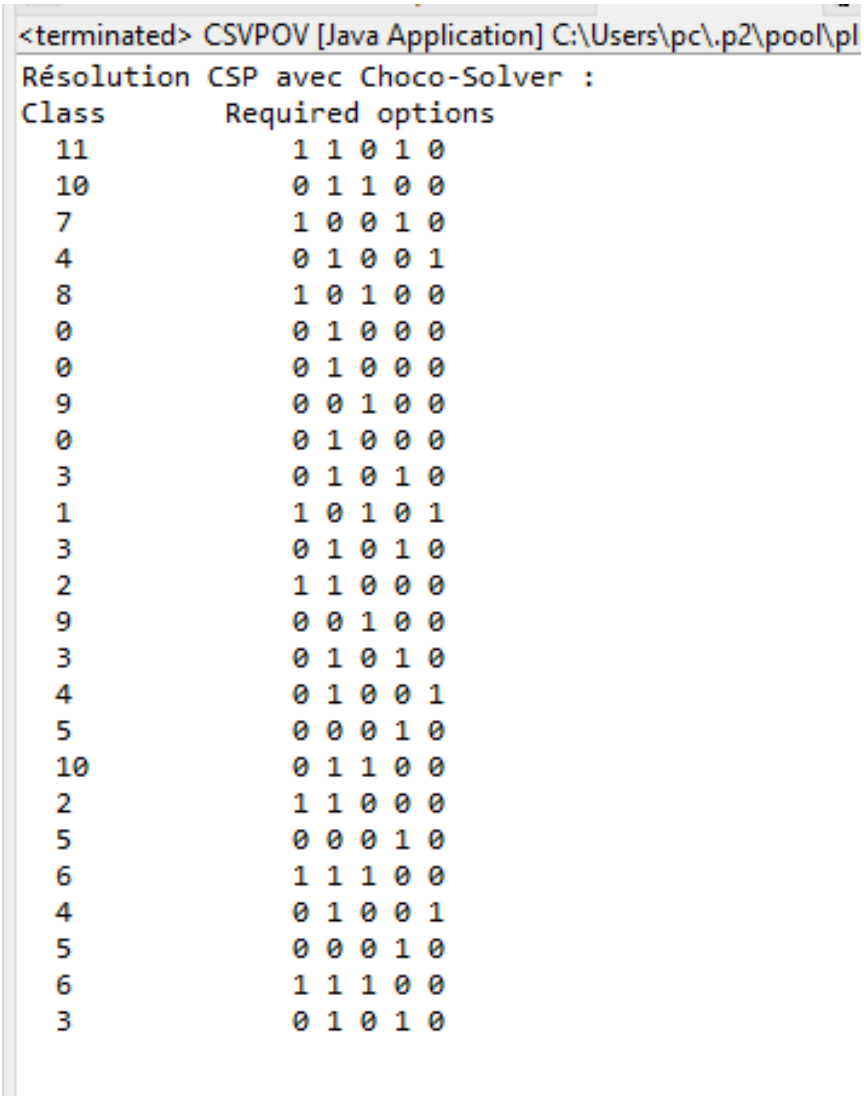
Cette contrainte implique que chaque séquence de q voiture doit avoir un maximum de p voiture pour chaque option i.

```
    public void SeqConstraint() {
        for(int i = 0; i < this.o; i++) {
            for(int j = 0; j < this.v-q[i]+1; j++) {
                IntegerExpressionVariable somme = ZERO ;
                for(int c = 0; c < q[i]; c++) {
                    somme = Choco.plus(somme,this.optionV[i][j+c]);
                }
                model.addConstraint(Choco.leq(somme,p[i]));
            }
        }
    }
```

Les fonctionnalités principales offertes par l'interface Solver sont la lecture du Model et la configuration d'un algorithme de recherche.

```
    public void solve() {
        this.solver.read(model);

        this.solver.solve();



            System.out.println("Résolution CSP avec Choco-Solver : ");

            System.out.println("Class  \t    Required options");
            for (int i = 0; i < this.v; i++) {
                System.out.print("  "+(solver.getVar(this.categoV[i]).getVal() - 1)+"\t   \t");
                for (int j = 0; j < this.o; j++) {
                    System.out.print(solver.getVar(this.optionV[j][i]).getVal() +" ");
                }
                System.out.println("");
            }
            System.out.println("");


}

    public static void main(String[] args) {

        new CSVPOV();
}
}
```

Les résultats après l'exécution :



- **Pour le nombre de voitures = 10 :**

module-info.java | CSVPOV.java

```java
package chocopov;

import choco.Choco;
import choco.cp.model.CPModel;
import choco.cp.solver.CPSolver;
import choco.kernel.model.Model;
import choco.kernel.model.constraints.Constraint;
import choco.kernel.model.variables.integer.IntegerExpressionVariable;
import choco.kernel.model.variables.integer.IntegerVariable;
import choco.kernel.solver.Solver;
import static choco.Choco.*;

public class CSVPOV {

    public CSVPOV() {
        this.init();
        this.NbreVConstraint();
        this.OptionConstraint();
        this.SeqConstraint();
        this.solve();
    }

    int v = 10; // Nombre de voitures à produire
    int o = 5; // Nombre des options disponibles
    int c = 6; // Nombre de classes

    // Listes d'entier pour chaque option i; p[i]/q[i] = la fréquence d'apparition
    int[] p = {1,2,1,2,1};
    int[] q = {2,3,3,5,5};

    int[] d = {1, 1, 2, 2, 2, 2}; // demande
    int[][] r = {
            {1, 0, 1, 1, 0},
            {0, 0, 0, 1, 0},
            {0, 1, 0, 0, 1},
            {0, 1, 0, 1, 0},
            {1, 0, 1, 0, 0},
            {1, 1, 0, 0, 0},
    }; //  Matrice de booléen ri,j = 1, représente le fait que l'option Oj est présente sur le véhicule Vi, 0 sinon.

```

Les résultats :

Console

<terminated> CSVPOV [Java Application] C:\Users\pc\.p2\pool\plugins\o

```
Résolution CSP avec Choco-Solver :
Class       Required options
  0             1 0 1 1 0
  1             0 0 0 1 0
  5             1 1 0 0 0
  2             0 1 0 0 1
  4             1 0 1 0 0
  3             0 1 0 1 0
  3             0 1 0 1 0
  4             1 0 1 0 0
  2             0 1 0 0 1
  5             1 1 0 0 0
```