ROYAUME DU MAROC

HAUT COMMISSARIAT AU PLAN

INSTITUT **N**ATIONAL
DE **S**TATISTIQUE ET D'**É**CONOMIE **A**PPLIQUÉE

Rapport projet POV

Réaliser par :

Abdelilah SABRI

Zakaria ALOUANI

M2SI – INSEA Rabat

Introduction

Le Problème d'Ordonnancement de Voitures (POV), mieux connu sous le nom de *Car-Sequencing Problem*, est apparu dans le secteur de la production automobile lorsque les fabricants ont abandonné la production de masse pour adopter la personnalisation des véhicules. Le POV consiste à donner l'ordre dans lequel des automobiles doivent être produites, en considérant les options de chacune et les contraintes de capacité de la chaîne de montage.

On peut le formuler en un problème de satisfaction de contraintes (CSP), dans ce rapport on va présenter une solution du problème en utilisant la bibliothèque **Choco** sous Java.

Choco est une bibliothèque Java libre et open-source dédiée à la programmation par contraintes dont l'utilisateur modélise son problème de manière déclarative en énonçant l'ensemble des contraintes qui doivent être satisfaites dans chaque solution. Ensuite, le problème est résolu en alternant des algorithmes de filtrage de contraintes avec un mécanisme de recherche.

Modélisation du problème

Définition des contraintes

Après une étude approfondie du problème **POV** nous avons déterminé trois contraintes qui sont comme suit :

- Contrainte qui précise le nombre de voiture demandé par chaque catégorie.
- Contrainte qui précise les options de chaque voiture selon sa catégorie.
- Contrainte qui précise le nombre maximum de voitures possédant une option qui peuvent être produites sur une sous-séquence de la chaîne de production.

Implémentation par Choco Solver

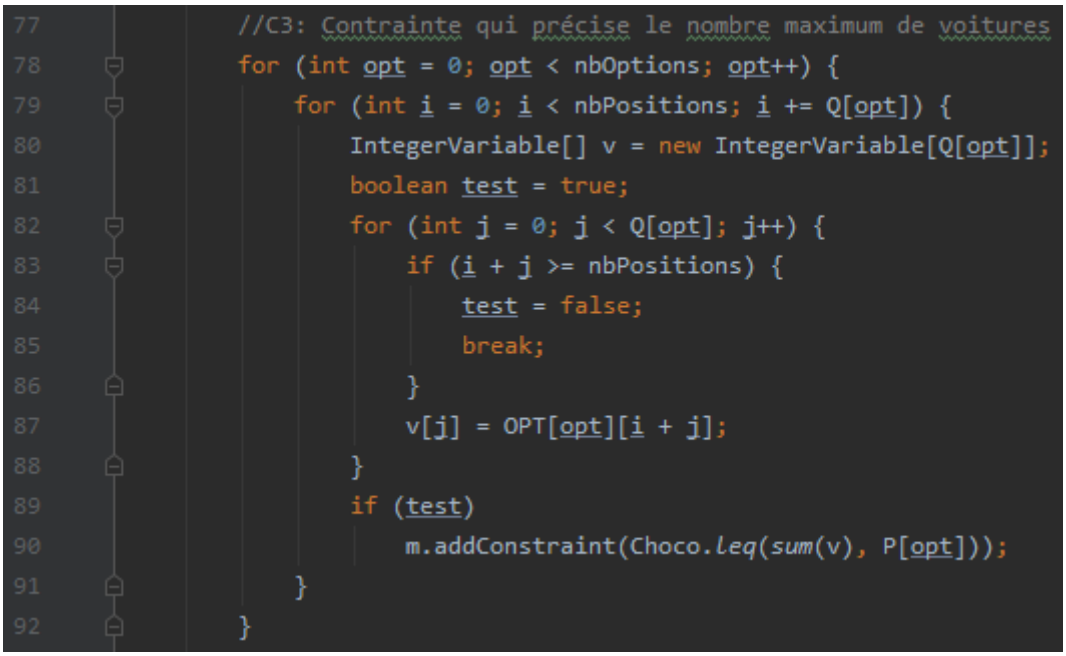
C1 : Contrainte qui précise le nombre de voiture demandé par chaque catégorie.

```
57      //C1: Contrainte qui precise le nombre de voiture demandé par chaque catégorie.
58      int[] classes = new int[nbCategories];
59      IntegerVariable[] classDemand = new IntegerVariable[nbCategories];
60      for (int j = 0; j < nbCategories; j++) {
61          classes[j] = j;
62          classDemand[j] = makeIntVar( name: "classDemand[" + j + "]", D[j], D[j]);
63      }
64      m.addConstraint(globalCardinality(S, classes, classDemand));
```

C2 : Contrainte qui précise les options de chaque voiture selon sa catégorie.

```
66      //C2: Contrainte qui précise les options de chaque voiture selon sa catégorie.
67      for (int cat = 0; cat < nbCategories; cat++) {
68          for (int car = 0; car < nbPositions; car++) {
69              Constraint[] C = new Constraint[nbOptions];
70              for (int op = 0; op < nbOptions; op++)
71                  C[op] = eq(OPT[op][car], options[cat][op]);
72
73              m.addConstraint(ifOnlyIf(and(C), eq(S[car], cat)));
74          }
75      }
```

C3 : Contrainte qui précise le nombre maximum de voitures possédant une option qui peuvent être produites sur une sous-séquence de la chaîne de production.



```
77          //C3: Contrainte qui précise le nombre maximum de voitures
78          for (int opt = 0; opt < nbOptions; opt++) {
79              for (int i = 0; i < nbPositions; i += Q[opt]) {
80                  IntegerVariable[] v = new IntegerVariable[Q[opt]];
81                  boolean test = true;
82                  for (int j = 0; j < Q[opt]; j++) {
83                      if (i + j >= nbPositions) {
84                          test = false;
85                          break;
86                      }
87                      v[j] = OPT[opt][i + j];
88                  }
89                  if (test)
90                      m.addConstraint(Choco.leq(sum(v), P[opt]));
91              }
92          }
```

Solution

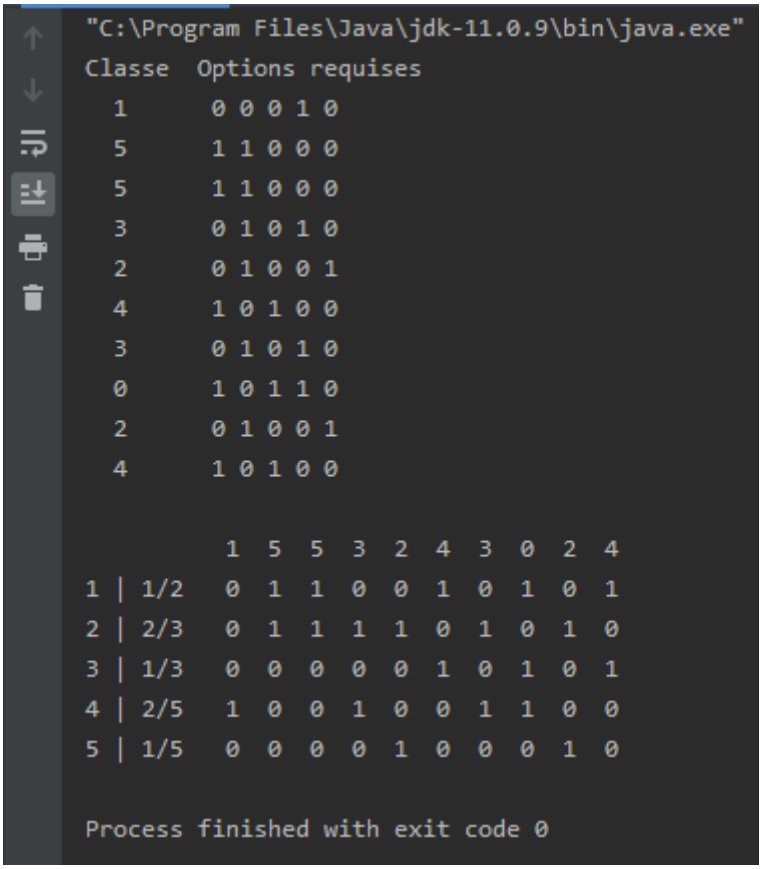
Définition des variables

```java
public static void main(String[] args) {
    // Nombre des options
    int nbOptions = 5;

    // Nombre des categories
    int nbCategories = 6;

    // Nombre total des voiture a produire
    int nbPositions = 10;

    // nombre maximum de voitures P / une sous-séquence de taille Q
    int[] P = {1, 2, 1, 2, 1};
    int[] Q= {2, 3, 3, 5, 5};

    // Nombre de voiture demandé par categorie.
    int[] D = {1, 1, 2, 2, 2, 2};

    //Les options de chaque categorie
    int[][] options =
            {
                    {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}
            };
```

Solveur et affichage des résultats

```
94          //Solveur
95          CPSolver s = new CPSolver();
96          s.read(m);
97          s.solve();
98
99          //L'affichage de la solution
100         System.out.println("Classe  Options requises");
101         for (int p = 0; p < nbPositions; p++) {
102             System.out.print("   " + s.getVar(S[p]).getVal() + "\t  \t ");
103             for (int c = 0; c < nbOptions; c++)
104             {
105                 System.out.print(s.getVar(OPT[c][p]).getVal() + " ");
106             }
107             System.out.println("");
108         }
109
110         System.out.println();
111
112         for (int p = 0; p < nbPositions; p++) {
113             if(p==0){
114                 System.out.print("\t\t  ");
115             }
116             System.out.print(s.getVar(S[p]).getVal() +"  ");
117         }
118         System.out.println();
119
120         for(int b = 0; b < nbOptions; b++){
121             System.out.print((b+1) + " | " + P[b] + "/" + Q[b] + "    ");
122             for (int p = 0; p < nbPositions; p++) {
123                 System.out.print(s.getVar(OPT[b][p]).getVal() + "  ");
124             }
125             System.out.println();
126         }
127     }
```

Les résultats



```
"C:\Program Files\Java\jdk-11.0.9\bin\java.exe"
Classe  Options requises
  1      0 0 0 1 0
  5      1 1 0 0 0
  5      1 1 0 0 0
  3      0 1 0 1 0
  2      0 1 0 0 1
  4      1 0 1 0 0
  3      0 1 0 1 0
  0      1 0 1 1 0
  2      0 1 0 0 1
  4      1 0 1 0 0

          1  5  5  3  2  4  3  0  2  4
1 | 1/2   0  1  1  0  0  1  0  1  0  1
2 | 2/3   0  1  1  1  1  0  1  0  1  0
3 | 1/3   0  0  0  0  0  1  0  1  0  1
4 | 2/5   1  0  0  1  0  0  1  1  0  0
5 | 1/5   0  0  0  0  1  0  0  0  1  0

Process finished with exit code 0
```

Conclusion

Le présent projet en programmation par contraintes (CSP), nous a permis d'utiliser le solveur **Choco** pour trouver la solution du problème d'ordonnancement des voitures, et cela en passant par une modélisation mathématique, ensuite une implémentation par choco.

A partir de ce projet nous avons vu l'importance et la valeur ajoutée de ce champ de programmation, plus spécialement dans des problèmes dont la programmation classique ne permet pas de trouver une solution au problème.