UNIVERSITÉ LIBRE DE BRUXELLES,
UNIVERSITÉ D'EUROPE

ULB

UNIVERSITÉ LIBRE DE BRUXELLES
FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

INFO-F-302 Informatique Fondamentale : Rapport de projet

PERALE Thomas
REQUENA Carlos

Année académique 2016 - 2017

Table des matières

1 Introduction

Le but du projet est de modéliser des problèmes de satisfaction de contraintes à l'aide de l'outil de résolution de contraintes `ChocoSolver` [1]. Pour celà on nous présente plusieurs problèmes en un premier temps basé sur un échiquier et la modélisation des pièces de celui-ci. En un deuxième temps basé sur la disposition d'un musée et de l'emplacement que doivent avoir les cameras dans celui-ci.

2 Question 1 : Le problème d'indépendance

2.1 Définition du problème

Comme il a été dit dans l'énoncé le problème d'indépendance consiste à déterminer dans un échiquier de taille donné si il est possible d'assigner à chacune des pièces misent à notre disposition (c'est à dire autant de chevalier, tour ou fou que l'on veut) une position distincte de sorte qu'aucune pièce ne menace une autre pièce.

2.2 Définition des variables

— n la taille de l'échiquier ;
— t le nombre de tour ;
— f le nombre de fou ;
— c le nombre de chevalier ;

2.3 Contraintes

— Chaque pièce doit occuper une case diffèrente.
— Aucune pièce ne peut occuper les cases situées à gauche, à droite, en haut et en bas des toutes les tours.
— Aucune pièce ne peut occuper les cases situées en diagonale des tous les fous.
— Aucune pièce ne peut occuper les cases menacées par tous les cavaliers (2.3)

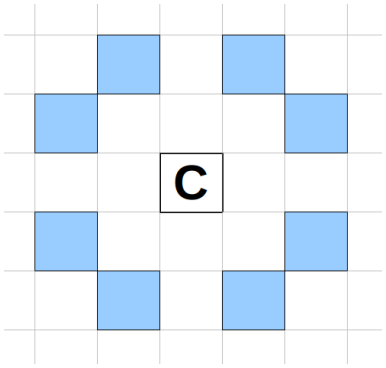

FIGURE 1 – Cases menacées par un cavalier

1. http ://www.choco-solver.org

3 Question 2 : Le problème de domination

3.1 Définition du problème

Ce problème cherche à ce que chaque case de l'échiquier doive être soit occupée soit menacée par au moins une pièce.

3.2 Définition des variables

- n la taille de l'échiquier ;
- t le nombre de tour ;
- f le nombre de fou ;
- c le nombre de chevalier ;

3.3 Contraintes

- Chaque pièce doit occuper une case diffèrente.
- Toute pièce doit occuper soit :
  - les cases situées à gauche, à droite, en haut et en bas des toutes les tours.
  - les cases situées en diagonale des tous les fous.
  - les cases menacées par tous les cavaliers (2.3).

4 Question 4 : Les chevaliers minimum

4.1 Définition du problème

Ce problème calcule le nombre minimal de cavaliers permettant de dominer un échiquier de taille donné.

Pour résoudre ce problème, il faut changer de tactique avec ChocoSolver. On ne cherche pas a savoir si une solution existe, mais plutôt a connaître la quantité minimale de chevaliers nécessaires pour menacer tout l'échiquier :

```
1 // to minimise X
2 model.setObjectives(Model.MINIMIZE, X);
```

4.2 Définition des variables

- n la taille de l'échiquier ;

4.3 Contraintes

- Chaque pièce doit occuper une case diffèrente. Chaque chevalier doit occuper une et une seule case libre de l'échiquier.
- Toutes les cases de l'échiquier doivent menacées par au moins un cavalier. C'est à dire, toutes les cases devront être bleues dans la figure (2.3), sauf les cases déjà occupées par les cavaliers.

5 Question 5 : La surveillance de musée

Ici le problème est essentiellement le même, appliqué à la surveillance d'un musée.

A Code Listing

```
package chocolatte;

import chocolatte.Domination;

import org.chocosolver.solver.Model;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.Solution;
import org.chocosolver.solver.search.strategy.Search;
import org.chocosolver.solver.variables.IntVar;

import com.google.common.collect.HashBasedTable;
import com.google.common.collect.Table;

// Argparse
import net.sourceforge.argparse4j.ArgumentParsers;
import net.sourceforge.argparse4j.inf.ArgumentParser;
import net.sourceforge.argparse4j.inf.ArgumentParserException;
import net.sourceforge.argparse4j.inf.Namespace;
import net.sourceforge.argparse4j.inf.MutuallyExclusiveGroup;
import net.sourceforge.argparse4j.impl.Arguments;


public class App {

    public static void main(String[] args) {
        ArgumentParser parser = ArgumentParsers.newArgumentParser("Chess pieces parser")
            .defaultHelp(true)
            .description("Returns a solution for the independence or domination problem for the given pieces (only rooks, knights and bishops)");
        MutuallyExclusiveGroup indom = parser.addMutuallyExclusiveGroup("Independence/Domination");
        indom.addArgument("-i").action(Arguments.storeTrue())
            .help("Solve the independence problem");
        indom.addArgument("-d").action(Arguments.storeTrue())
            .help("Solve the domination problem");
        indom.addArgument("-mk").action(Arguments.storeTrue())
            .help("Solve the minimum knights problem");
        parser.addArgument("-n")
            .type(Integer.class)
            .help("Generate a chess board NxN")
            .setDefault(5);
        parser.addArgument("-t")
            .type(Integer.class)
            .help("Number of rooks to place on board")
            .setDefault(2);
        parser.addArgument("-f")
            .type(Integer.class)
            .help("Number of bishops to place on board")
            .setDefault(2);
        parser.addArgument("-c")
            .type(Integer.class)
            .help("Number of knights to place on board")
            .setDefault(2);
        parser.epilog("Usage with Gradle: \n$ gradle run -PappArgs=\"['-flag1', '-arg2 x', 'option3']\"");
        Namespace ns = null;
        try {
            ns = parser.parseArgs(args); // Fill the namespace with pass arguments
        } catch (ArgumentParserException e) {
            parser.handleError(e);
            System.exit(1);
        }
```

```
60
61          App app = new App();
62
63          boolean domination = ns.getBoolean("d");
64          boolean minimum_knights = ns.getBoolean("mk");
65          int boardSize = ns.getInt("n");
66          int rook = ns.getInt("t");
67          int bishop = ns.getInt("f");
68          int knight = ns.getInt("c");
69
70          Board chessBoard = new Board(boardSize, rook, bishop, knight);
71          Solution sol = null;
72
73          if (minimum_knights) {
74              chessBoard.createPotentialKnights();
75              MinimalKnights min = new MinimalKnights(chessBoard);
76              sol = min.exec();
77              chessBoard.printKnightsBoard(sol);
78          } else if (domination) {
79              chessBoard.createPieces();
80              Domination dom = new Domination(chessBoard);
81              sol = dom.exec();
82              chessBoard.printSolutionBoard(sol);
83          } else {
84              chessBoard.createPieces();
85              Independence ind = new Independence(chessBoard);
86              sol = ind.exec();
87              chessBoard.printSolutionBoard(sol);
88          }
89      }
90  }
```