# Modélisation et résolution pour l'optimisation

Noms

November 2, 2024

## 1 Problèmes d'optimisation sous contraintes

### 1.1 Modélisation du problème

(i)

$x_{ij} :=$ Fréquence de station $i$ à station $j$
$t_i :=$ Région de station $i$

$X = \{x_{11}, \ldots, x_{nn}\} \cup \{r_1, \ldots, r_n\}$
$D = \{d_{x_{11}}, \ldots d_{x_{nn}}\} \cup \{d_{r_1}, \ldots, d_{r_n}\}$ avec pour tout $i, j : d_{x_{ij}} = \mathbb{N}, r_i = \{1, \ldots, k\}$
$C = \{\forall i \in \{1, \ldots, n\} : \exists j, k \neq i \Rightarrow \delta_i = |x_{ij} - x_{ji}|\}$
$\cup \{\forall x_{ij} \in X : \exists k \neq i, l \neq j \Rightarrow |x_{ij} - x_{jl}| \geq \Delta_{ij}, |x_{ij} - x_{ki}| \geq \Delta_{ij}\}$

- $\min |X|$
- $\min\limits_{x_i \in X} x_i$
- $\min\ (\max\limits_{x_i \in X} x_i - \min\limits_{x_i \in X} x_i)$

### 1.2 Instance XCSP3

Le code COP.py (cf annexe) nous permet de générer une instance XCSP3 à partir de nos données. Ainsi partant du jeu de données data.json suivant :

```
{
    "stations": [
            { "num":  0, "region":  0, "delta": 140, "emetteur": [ 14, 28, 42, 56, 7
            { "num":  1, "region":  1, "delta": 140, "emetteur": [ 14, 28, 42, 56, 7
            { "num":  2, "region":  1, "delta": 140, "emetteur": [ 14, 28, 42, 56, 7
```

```
    ],
    "regions": [2, 3],
    "interferences": [
            { "x": 0, "y": 1, "Delta": 20 },
            { "x": 0, "y": 2, "Delta": 30 }
    ],
    "liaisons": [
            { "x": 1, "y": 2 }
    ]
}
```

Nous obtenons l'instance XCSP3 suivante :

```
<instance format="XCSP3" type="CSP">
<variables>
    <array id="fe" note="Variables pour les fréquences d'émission et de réception po
    <domain for="fe[0] fe[1]"> 14 28 42 56 70 84 98 112 126 140 154 168 182 196 </do
    <domain for="fe[2]"> 14 28 42 56 70 84 98 112 126 140 154 168 196 </domain>
    </array>
    <array id="fr" size="[3]">
    <domain for="fr[0]"> 14 28 42 56 70 126 140 154 182 196 </domain>
    <domain for="fr[1]"> 14 28 42 56 70 98 140 154 168 182 196 </domain>
    <domain for="fr[2]"> 14 28 42 56 70 84 98 112 126 140 154 168 182 196 </domain>
    </array>
</variables>
<constraints>
    <intension> eq(dist(fe[0],fr[0]),140) </intension>
    <intension> eq(dist(fe[1],fr[1]),140) </intension>
    <intension> eq(dist(fe[2],fr[2]),140) </intension>
    <intension> ge(dist(fe[0],fe[1]),20) </intension>
    <intension> ge(dist(fr[0],fr[1]),20) </intension>
    <intension> ge(dist(fe[0],fe[2]),30) </intension>
    <intension> ge(dist(fr[0],fr[2]),30) </intension>
    <nValues>
    <list> fe[0] fr[0] </list>
    <condition> (le,2) </condition>
    </nValues>
    <nValues>
    <list> fe[1] fe[2] fr[1] fr[2] </list>
    <condition> (le,3) </condition>
    </nValues>
</constraints>
</instance>
```

## 1.3 Comparaison des différents solveurs

Dans cette partie nous allons évaluer les performances des différents solveurs suivants : "Choco", "MiniZinc", "Gecode", "CBC" et "Google OR-Tools". Ces évaluations seront basées sur une comparaison du temps d'exécution ainsi que sur une mesure de performance que nous expliciterons dans la partie dédiée.

### 1.3.1 Comparaison en temps

### 1.3.2 Comparaion des performances

# 2 Problèmes de satisfaction de contraintes valuées

## 2.1 Modélisation du problème

## 2.2 Instance WSCP

## 2.3 Résolution avec le solveur Toulbar2