Modélisation et résolution pour l'optimisation

Bouger Lisa, Yannick Brenning, Baptiste Chachura

December 8, 2024

1 Problèmes d'optimisation sous contraintes

1.1 Modélisation du problème

1. Variables et définitions :

   x_{ij} := Fréquence entre la station i et la station j,
   t_i := Région associée à la 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 : $d_{x_{ij}} = \mathbb{N},\ d_{r_i} = \{1, \ldots, k\}$,
   $C = \{\forall i \in \{1, \ldots, n\}, \exists j, k \neq i \Rightarrow \delta_i = |x_{ij} - x_{ji}|\}$
   $\cup\ \{\forall x_{ij} \in X, \exists k \neq i, l \neq j \Rightarrow |x_{ij} - x_{jl}| \geq \Delta_{ij},\ |x_{ij} - x_{ki}| \geq \Delta_{ij}\}$.

2. Fonction(s) objectif :

   - Minimiser $|X|$
     On cherche à minimiser le nombre de fréquences utilisées.
   - minimiser $\underset{x_i \in X}{x_i}$
     On cherche à utiliser les fréquences les plus petites possibles.
   - Minimiser $\left(\max_{x_i \in X} x_i - \min_{x_i \in X} x_i\right)$
     On cherche à minimiser la bande passante utilisée.

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, \\
        70, 84, 98, 112, 126, 140, 154, 168, 182, 196 ],
          "recepteur": [ 14, 28, 42, 56, 70, 126, 140, 154, 182, 196 ] },
        { "num":  1, "region":  1, "delta": 140, "emetteur": [ 14, 28, 42, 56, \\
        70, 84, 98, 112, 126, 140, 154, 168, 182, 196 ],
          "recepteur": [ 14, 28, 42, 56, 70, 98, 140, 154, 168, 182, 196 ] },
        { "num":  2, "region":  1, "delta": 140, "emetteur": [ 14, 28, 42, 56, \\
        70, 84, 98, 112, 126, 140, 154, 168, 196 ],
          "recepteur": [ 14, 28, 42, 56, 70, 84, 98, 112, 126, 140, 154, 168, 182, 196 ]
    ],
    "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 pour chaque station" size="[3]">
            <domain for="fe[0] fe[1]">
                14 28 42 56 70 84 98 112 126 140 154 168 182 196
            </domain>
            <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 disponibles dans le package pycsp3 : "Choco" et "ACE". 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

Dans cette partie les numéros assignés aux objectifs correspondent aux trois objectifs décrit en partie 1 :

- 1 : minimiser le nombre de fréquences utilisées
- 2: minimiser les valeurs des fréquences utilisées
- 3: minimiser la largeur de la bande passante

Nous constatons que les performances varient en fonction de l'objectif à optimiser néanmoins le solveur choco montre de bien meilleur performances en terme de temps de résolution que le solveur ACE. De plus, lors de minimisation des valeurs des fréquences utilisées nous observons que le solveur ACE perfome en un temps de 60 secondes, cela est dû à une condition de notre code qui impose un timeout à 60 secondes. En effet lors de nos tests nous avons constaté que sous cette condition le solveur ACE ne convergeait pas dans un temps acceptable. (pas de convergence en 8h30 d'exécution pour un fichier de 50 stations)

Objectif	Nb Station	Time ACE	Time Choco
1	50	0.725848	0.117340
1	150	1.149676	0.118492
1	250	2.005245	0.118255
1	500	4.645630	0.134006
2	50	60.341584	0.109806
2	150	60.349703	0.119203
2	250	60.360382	0.122928
2	500	60.389112	0.135731
3	50	0.587273	0.114264
3	150	0.742485	0.123299
3	250	0.994855	0.126541
3	500	1.402266	0.136405

Table 1: Résultats des temps pour ACE et Choco en fonction du nombre de stations et de l'objectif.

1.3.2 Comparaison des performances

1. **Objectif 1 : Minimiser le nombre de fréquences utilisées**
   Pour cet objectif, nous observons que les deux solveurs atteignent des performances équivalentes. Sur l'ensemble des fichiers de test, ils parviennent à déterminer une configuration nécessitant uniquement quatre fréquences. De plus, pour chaque instance, les deux solveurs convergent systématiquement vers le même jeu de fréquences. Cela témoigne de leur efficacité et de leur cohérence dans la résolution de ce type de problème.

2. **Objectif 2 : Utiliser les fréquences les plus basses possibles**
   Pour évaluer cette performance, nous avons utilisé le même critère que lors de l'exécution du script : minimiser la somme des fréquences utilisées. Les résultats obtenus sont néanmoins surprenants. En effet, lors de la résolution de cet objectif, le solveur ACE atteignait systématiquement le timeout, tandis que Choco parvenait à résoudre le problème en un temps quasi constant. Nous nous attendions à observer de grandes différences de performance ; cependant, ce n'est pas le cas. Pour toutes les instances de test, les deux solveurs produisent les mêmes résultats finaux.

3. **Objectif 3 : Minimiser la largeur de la bande passante**
   Encore une fois pour chacune des instances, les deux solveurs conduisent aux mêmes résultats.

1.4 Conclusion

Pour conclure cette première partie dédiée à la résolution de problèmes sous contraintes, nous pouvons établir les points suivants :

1. **Simplicité et flexibilité de l'API PyCSP3** L'implémentation Python utilisant l'API PyCSP3 permet de modéliser aisément une variété de problèmes et de contraintes. Elle offre également la possibilité de tester différents solveurs en les intégrant directement au sein du code, ce qui représente un atout majeur en termes de flexibilité et d'adaptabilité.

2. **Résultats équivalents des solveurs** Dans le cadre de notre problème, les deux solveurs testés, `ACE` et `Choco`, génèrent systématiquement les mêmes solutions. Cette convergence peut probablement être attribuée à leur approche commune basée sur la stratégie de *branch and bound*. Cependant, ces résultats ne permettent pas de juger des performances respectives de ces solveurs en termes d'optimisation des critères définis, car ils aboutissent tous deux aux mêmes résultats.

3. **Avantage temporel de `Choco`** Un avantage significatif est constaté pour le solveur `Choco`, qui atteint la solution en un temps quasi constant pour tous les objectifs. À l'inverse, le temps de résolution avec `ACE` dépend de la "taille" du problème initial. Cette différence rend `Choco` potentiellement plus adapté à des problèmes de plus grande dimension ou à des cas nécessitant une résolution rapide.

2 Problèmes de satisfaction de contraintes valuées

2.1 Modélisation du problème

2.2 Instance WSCP

2.3 Résolution avec le solveur Toulbar2