AI09
Programmation par Contraintes - Exercice 11

Professeur : Savourey David
Étudiants : Pereira Hugo & Zizouni Maher

16 décembre 2024

Table des matières

1 Introduction

Ce rapport présente notre solution à l'exercice 11 du cours de Programmation par Contraintes. L'objectif de cet exercice est de résoudre un problème d'ordonnancement utilisant la programmation par contraintes. Nous verrons dans un premier temps comment écrire un CSP pour ce problème, puis nous présenterons notre solution développée en GNU Prolog.

2 Modélisation du problème

On définira donc un CSP par un triplet (X, D, C) tel que vu en cours.

2.1 Variables

- $X = \{DEBUT_1, DEBUT_2, ..., DEBUT_i\}$: la date de début de la tâche i.

2.2 Domaines

- $D(DEBUT_i) = \{R_i, R_i + 1, ..., D_i - P_i\}$: car la tâche i commence après sa date de disponibilité ET avant sa date échue moins sa durée.

2.3 Contraintes

- Les contraintes de début et de fin de chaque tâche sont assurées par le domaine.
- $C = \{DEBUT_i + P_i \leq DEBUT_j$ ou $DEBUT_j + P_j \leq DEBUT_i\}$: les tâches i et j avec $i \neq j$ ne peuvent pas se chevaucher car la machine traite une tâche après l'autre.

3 Application du CSP en GNU Prolog

Le CSP sera appliqué sur le problème suivant :

```
R1 #= 3, P1 #= 5, D1 #= 18,
R2 #= 1, P2 #= 6, D2 #= 10,
R3 #= 5, P3 #= 2, D3 #= 14,
R4 #= 4, P4 #= 1, D4 #= 12,
R5 #= 0, P5 #= 4, D5 #= 22,
```

FIGURE 1 – Présentation des données du problème

3.1 Variables et domaines

On a donc les variables $DEBUT_i$ pour chaque tâche $i \in [1, 5]$ et les domaines comme énoncés précédemment dans notre CSP.

```
D1_P1 is D1 - P1,
D2_P2 is D2 - P2,
D3_P3 is D3 - P3,
D4_P4 is D4 - P4,
D5_P5 is D5 - P5,
fd_domain([DEBUT_1], R1, D1_P1),
fd_domain([DEBUT_2], R2, D2_P2),
fd_domain([DEBUT_3], R3, D3_P3),
fd_domain([DEBUT_4], R4, D4_P4),
fd_domain([DEBUT_5], R5, D5_P5),
```

FIGURE 2 – Variables et domaines

3.2 Contraintes

On a donc les contraintes de chevauchement des tâches comme énoncées précédemment dans notre CSP.

```
(DEBUT_1+P1 #=< DEBUT_2) #\/ (DEBUT_2+P2 #=< DEBUT_1),
(DEBUT_1+P1 #=< DEBUT_3) #\/ (DEBUT_3+P3 #=< DEBUT_1),
(DEBUT_1+P1 #=< DEBUT_4) #\/ (DEBUT_4+P4 #=< DEBUT_1),
(DEBUT_1+P1 #=< DEBUT_5) #\/ (DEBUT_5+P5 #=< DEBUT_1),
(DEBUT_2+P2 #=< DEBUT_3) #\/ (DEBUT_3+P3 #=< DEBUT_2),
(DEBUT_2+P2 #=< DEBUT_4) #\/ (DEBUT_4+P4 #=< DEBUT_2),
(DEBUT_2+P2 #=< DEBUT_5) #\/ (DEBUT_5+P5 #=< DEBUT_2),
(DEBUT_3+P3 #=< DEBUT_4) #\/ (DEBUT_4+P4 #=< DEBUT_3),
(DEBUT_3+P3 #=< DEBUT_5) #\/ (DEBUT_5+P5 #=< DEBUT_3),
(DEBUT_4+P4 #=< DEBUT_5) #\/ (DEBUT_5+P5 #=< DEBUT_4),
```

FIGURE 3 – Contraintes de non chevauchement

3.3 Résolution et résultats

Pour résoudre le problème, on utilise la fonction `fd_labeling` de GNU Prolog.

```
1 fd_labeling([DEBUT_1, DEBUT_2, DEBUT_3, DEBUT_4, DEBUT_5]),
```

FIGURE 4 – Résolution du problème

On affiche ensuite les résultats.

```
1  nl,
2  write('DEBUT_1='), write(DEBUT_1),
3  nl,
4  write('DEBUT_2='), write(DEBUT_2),
5  nl,
6  write('DEBUT_3='), write(DEBUT_3),
7  nl,
8  write('DEBUT_4='), write(DEBUT_4),
9  nl,
10 write('DEBUT_5='), write(DEBUT_5),
11 nl.
```

FIGURE 5 – Affichage des résultats

Et on obtient des résultats qui respectent les contraintes. Notre solveur trouve plusieurs solutions, ces dernières se trouvent dans le fichier `dev/resultats/resultats_sans_optimiser.md`.

4 Optimisation du problème

4.1 Optimisation de la date de fin

Si on cherche à optimiser la date de fin du projet, on peut ajouter une variable FIN qui représente la date de fin du projet et une contrainte qui lie cette variable aux dates de fin des tâches. On pourra ensuite utiliser la fonction `fd_minimize` de GNU Prolog pour minimiser cette variable. Il ne nous semble pas nécessaire de renseigner le domaine de cette nouvelle variable puisque les autres contraintes lui donneront de fait une borne inférieure. Cependant, après certaines recherches sur internet, il semble que cela soit nécessaire pour permettre au solveur d'effectuer la propagation et d'explorer la solution de manière efficace. On choisit 22 comme borne supérieure pour cette variable car c'est la date de fin maximale possible.

```
fd_domain([FIN], 0, 22),

DEBUT_1 + P1 #=< FIN,
DEBUT_2 + P2 #=< FIN,
DEBUT_3 + P3 #=< FIN,
DEBUT_4 + P4 #=< FIN,
DEBUT_5 + P5 #=< FIN,

fd_minimize(fd_labeling([DEBUT_1, DEBUT_2, DEBUT_3, DEBUT_4, DEBUT_5, FIN]), FIN),
```

FIGURE 6 – Optimisation de la date de fin

Cette fois, on obtient une seule solution optimale, qui est la suivante :

```
DEBUT_1=13
DEBUT_2=4
DEBUT_3=11
DEBUT_4=10
DEBUT_5=0
FIN=18
```

FIGURE 7 – Résultat de la première optimisation

4.2 Optimisation de la somme des dates de fin

Si on cherche à optimiser la somme des dates de fin des tâches, on peut ajouter une variable $SOMME_FIN$ qui représente la somme des dates de fin des tâches et une contrainte qui lie cette variable aux dates de fin des tâches. On pourra ensuite utiliser la fonction `fd_minimize` de GNU Prolog pour minimiser cette variable. On choisit 200 comme borne supérieure pour cette variable car c'est une limite supérieure suffisante pour couvrir toutes les solutions réalisables.

```
fd_domain([SOMME_FIN], 0, 200),

SOMME_FIN #= (DEBUT_1 + P1) + (DEBUT_2 + P2) + (DEBUT_3 + P3)
             + (DEBUT_4 + P4) + (DEBUT_5 + P5),

fd_minimize(fd_labeling([DEBUT_1, DEBUT_2, DEBUT_3, DEBUT_4, DEBUT_5,
            SOMME_FIN]), SOMME_FIN),

```

FIGURE 8 – Optimisation de la somme des dates de fin

On obtient également une seule solution optimale, qui est la suivante :

```
DEBUT_1=13
DEBUT_2=4
DEBUT_3=11
DEBUT_4=10
DEBUT_5=0
SOMME_FIN=56
```

FIGURE 9 – Résultat de la deuxième optimisation

5 Conclusion

Nous avons donc vu comment modéliser un problème d'ordonnancement en utilisant la programmation par contraintes. Nous avons ensuite implémenté notre solution en GNU Prolog et avons pu résoudre le problème de manière optimale en minimisant la date de fin du projet et la somme des dates de fin des tâches. Nous avons également pu observer que la solution optimale est la même pour les deux optimisations. Cela s'explique par le fait que toutes les tâches sont ordonnancées sans le moindre temps mort. Il n'y a donc pas de différence entre minimiser la date de fin du projet et minimiser la somme des dates de fin des tâches.

6 Sources

- **Cours de Programmation par Contraintes, UTC (AI09)** : Notes de cours fournies par le professeur.
- **Documentation de GNU Prolog** : Disponible sur `http://www.gprolog.org/`.
- Apté, P. (2001). *Introduction to Constraint Programming*. Springer.
- Rossi, F., Van Beek, P., & Walsh, T. (2006). *Handbook of Constraint Programming*. Elsevier.
- Apt, K.R. (2003). *Principles of Constraint Programming*. Cambridge University Press.
- Articles et discussions sur les forums Stack Overflow concernant les meilleures pratiques en programmation par contraintes et en GNU Prolog.

Table des figures