Approches par contraintes pour la résolution d'un problème d'ordonnancement

Ronan Bocquillon

Je vous propose une initiation à la programmation par contraintes au travers de cette étude de cas, portant sur un problème d'ordonnancement "académique" très étudié, le "$1|r_i|L_{max}$".

1 Description du problème

Le problème $1|r_i|L_{max}$, pour *one machine*, *release dates*, *maximum lateness*, s'énonce comme suit.

Soit $T = \{1, 2, ..., n\}$, un ensemble de n tâches, à ordonnancer sur une seule machine. Chaque tâche $i \in T$ est caractérisée par sa durée opératoire $p_i \in \mathbb{N}$, sa date de disponibilité $r_i \in \mathbb{N}$, et par sa date de fin souhaitée $dd_i \in \mathbb{N}$. La machine ne peut exécuter qu'une seule tâche à la fois ; une tâche ne peut pas commencer avant sa date de disponibilité ; et la préemption est interdite. Ainsi, lorsqu'une tâche commence, elle occupe la machine pendant toute sa durée opératoire, *sans interruption*.

On note C_i, la date de fin de la tâche $i \in T$. L'ordonnancement calculé doit minimiser $L_{max} = \max_{i \in T} (C_i - dd_i)$ (*maximum lateness*).

2 Travail préliminaire

On considère l'instance suivante :

i	r_i	p_i	dd_i
1	33	32	73
2	39	13	37
3	16	29	29
4	0	1	70
5	39	29	69

1. Dessinez les diagrammes de Gantt correspondant aux séquences de tâches $(1, 2, 3, 4, 5)$ et $(4, 3, 1, 5, 2)$.
2. Calculez le L_{max} de ces deux solutions.

3 Modélisation

Proposez une modélisation en contraintes du problème étudié. Pensez à bien définir vos variables, leur domaine, la fonction objectif et les contraintes.

4 Implémention

Vous allez maintenant tester votre modèle sur deux solveurs de contraintes éprouvés (Choco Solver et IBM ILOG CP Optimizer).

4.1 Utilisation de Choco Solver

Voici la démarche proposée :

1. Préparez un environnement de développement Java, puis créez et testez un projet Choco Solver simple.
   (a) `https://choco-solver.org/docs/getting-started/`
   (b) `https://choco-solver.org/tutos/first-example/`
2. Importez la classe `Instance` fournie. Générez et affichez des instances de petite taille ($n = 10$, $K = 15$).
3. Implémentez une classe `Activity`, qui encapsule les variables `start`, `end` et `proc` décrites dans le cours.
   (a) Javadoc de `IVariableFactory` (interface impl. par `Model`)
   (b) (optionnel) Javadoc de `IntOffsetView`
4. Inspirez vous des exemples de la documentation pour implémenter votre modèle (NB. pour la contrainte disjonctive, ajoutez ${}^{n(n-1)}/_{2}$ contraintes $end(A_i) \leq start(A_j) \vee end(A_j) \leq start(A_i)$). Testez la méthode.
5. (optionnel) Implémentez Edge-Finding (*cf.* Algorithme 4.1 ; vous pouvez vous contenter de l'algorithme primal pour commencer).
   (a) `https://choco-solver.org/docs/advanced-usages/propagator/`
6. (optionnel) Testez aussi différentes stratégies de branchement.
   (a) `https://choco-solver.org/docs/solving/strategies/`

Algorithm 1 Edge-Finding [Nuijten *et al.*, 1993]

Require: The set of activities is ordered by ascending release dates.

Require: This is the primal version of the algorithm, which updates the earliest start times only. It should be implemented along with its dual version, which updates the latest end times.

```
1:  for i := 1 to n do
2:     r'_i := r_i
3:  end for
4:  for k := 1 to n do
5:     P := 0, C := −∞, H := −∞
6:     for i := n down to 1 do
7:        if d_i ≤ d_k then
8:           P := P + p_i
9:           C := max(C, r_i + P)
10:          if C > d_k then
11:             backtrack (there is no feasible schedule)
12:          end if
13:       end if
14:       C_i := C
15:    end for
16:    for i := 1 to n do
17:       if d_i ≤ d_k then
18:          H := max(H, r_i + P)
19:          P := P − p_i
20:       else
21:          if r_i + P + p_i > d_k then
22:             r'_i := max(r'_i, C_i)
23:          end if
24:          if H + p_i > d_k then
25:             r'_i := max(r'_i, C)
26:          end if
27:       end if
28:    end for
29: end for
30: for i := 1 to n do
31:    r_i := r'_i
32: end for
```

4.2 Utilisation de IBM ILOG CP Optimizer

Voici la démarche proposée :

1. Lancez IBM ILOG CPLEX Optimization Studio.
2. Importez le projet exemple `sched_pflowshop`.
3. Inspirez vous en pour implémenter votre propre modèle.
4. Testez et comparez vos deux implémentations [1].

4.3 Pour aller plus loin...

Comme vous avez pu le constater, CP Optimizer propose de très nombreuses fonctionnalités (variables d'intervalle, contraintes de ressources, *etc.*) dédiées à la modélisation et à la résolution des problèmes d'ordonnancement. Il est donc particulièrement adapté à ces problèmes.

Si vous souhaitez aller plus loin, vous pouvez implémenter votre modèle avec l'API `C++` de CP Optimizer. Cela vous permettra de développer vos contraintes, vos stratégies de recherche, *etc.* Dans le cas présent, il pourrait par exemple être intéressant de développer une contrainte pour propager des bornes inférieures du L_{max}.

1. La méthode `write` de la classe `Instance` vous permettra aisément de générer des fichiers de données `.dat` depuis votre projet Java.