Resource Constrained Project Scheduling Problem (RCPSP)

BENCHEIKH Ghita et SADIK Meriem

18 Janvier 2016

Résumé

L'essentiel pour résoudre un problème est de trouver un bon modèle qui le décrit de façon réaliste et qu'il soit suffisamment simple, notons bien qu'un petit changement de modèle peut entraîner de grands changements de complexité. Dans ce travail nous traitons le problème d'ordonnancement avec des contraintes temporelles entre les tâches et des contraintes de ressources, et comme tout problème d'ordonnancement on souhaite minimiser la date de complétion d'exécution totale du projet. Nous présentons quelques modèles issus de la littérature, puis nous proposons quelques améliorations à ces modèles, en détectant les cas inconsistants et en calculant des bornes de la solution.

I. Position du problème

Dans ce travail, nous nous intéressons au problème d'ordonnancement de projet à moyens limités ou Resource Constrained Project Scheduling, qui consiste à ordonnancer un ensemble de tâches liées entre elles par des contraintes de précédences, en leurs attribuant la quantité de ressources dont chacune nécessite de chaque ressource.

Les tâches du projet sont identifiées par l'ensemble $V = \{A_0, A_1, ..., A_{n+1}\}$, l'activité A_0(resp. l'activité A_{n+1}) représente le début du projet(resp. la fin du projet), et $A = \{A_1, ..., A_n\}$ l'ensemble des activités à ordonnancer. Les durées des tâches sont représentés par un vecteur p de taille $\mathbb{N}^{n+2}$, et on écrit p_i la durée de l'activité i, sachant que $p_0 = p_{n+1} = 0$, et finalement l'ensemble des ressources renouvelables $R = \{R_1, ..., R_q\}$, avec B_k la quantité de ressources consommables de R_k à chaque instant.

On note :

- $G(V, E)$: graphe des relations de précédences, tel que si : $(A_i, A_j) \in E \iff A_i$ précède A_j (sans circuits)
- b_{ik} : ce que consomme l'activité A_i de la ressources R_k

Le problème consiste à trouver les dates de début des tâches $S = \{S_1, ..., S_n\}$ de telle sort que la date d'achèvement du projet soit minimale, tout en respectant les contraintes de ressources et de précédences.

Le problème peut se formuler comme suit :

$$(P) \begin{cases} min\ \mathrm{S}_{n+1} \\ sc. \\ \quad \mathrm{S}_i + p_i \leq \mathrm{S}_{n+1} \\ \quad \mathrm{S}_j - S_i \geq \mathrm{p}_i\ \forall (\mathrm{A}_i, A_j) \in E \\ \quad \sum_{A_i \in A_t} b_{ik} \leq \mathrm{B}_k\ \forall \mathrm{R}_k \in R, \forall t \geq 0 \\ \quad \mathrm{S}_i \in \mathbb{R}^+ \end{cases}$$

Avec A_t l'ensemble des activités ordonnancées pendant t.

II. Complexité du problème

Le RCPSP est un problème d'optimisation NP-difficile au sens fort, en raison de sa nature fortement combinatoire. Un problème combinatoire est NP-difficile au sens forte, si sa version décisionnelle est NP-complet.

II.1 version décisionnelle du problème RCPSP

Definition II.1 *Existe-t-il un ordonnancement S de valeur au plus H, respectant les contraintes des précedences et de ressources.*

II.1.1 Classe NP

D'abord, on vérifie que ce problème est dans la classe NP (existe-t-il un algorithme avec lequel on peut vérifier en temps polynomial si une solution S est réalisable ou pas). En effet, On peut facilement vérifier si les contraintes de précédences sont vérifiées en $o(|E|)$, par contre la vérification de la réalisabilité des contraintes de ressources, n'est pas trivial, mais il existe un algorithme(Algorithme1) qui le fait en temps pseudo-polynomiale de complexité $o(n^2\,|R|)$, l'algorithme détermine l'ensemble F des activités ordonnancées à l'instant $t \in L$, avec L l'ensemble des dates d'achèvement des tâches trié par ordre croissant, et vérifie si la somme des ressources consommées par ces activités de chaque ressource R_k ne dépasse pas la capacités de cette dernière($\sum_{i\in F} b_{ik} \leq B_k, \forall k \in R_k$).

Algorithm 1: Réalisabilité des contraintes de ressources

```
Data: Liste L = { C_j |A_j ∈ V }
Result: Vrai/Faux
Initislisation : trié la liste L par ordre décroissant.
for t ∈ L do
    for R_k ∈ R do
        o ⟵ 0, F ⟵ ∅
        for A_j ∈ A do
            if S_j < t, C_j ≥ t and b_jk > 0 then
                o ⟵ o + b_jk, F ⟵ F ∪ A_j
            end
            if o > B_k then
                S ne respecte pas la contrainte de ressource R_k
            end
            return Faux
        end
    end
end
return Vrai
```

II.1.2 NP-hard

- Sans précédences, on peut réduire le problème de 3-partition en ce problème[1],
- Le problème de coloration d'un graphe apparaît comme une variante du problème RCPSP sans contrainte de précédences et des durées et des quantités de ressources unitaires[3].
- Au contraire, sans contrainte de limitation de ressources, le problème devient devient un cas particulier du problème d'ordonnancement et se résout en temps polynomial.

III. Formulations mathématique

On distingue deux types de formulations, celles en temps continu, qui sont basées sur des variables binaires x_{ij} qui vaut 1 si la tâche A_i est exécutée avant la tâche A_j et 0 sinon, et celles en temps discrétisé basant sur des variables binaires y_{jt} qui vaut 1 si la tâche A_j a commencé l'exécution au temps t. Ainsi dans ce travail nous présentons 3 formulations basiques issues de la littérature.

III.1 Formulations en temps continu

III.1.1 Modèle de flot

En partant du principe que chaque unité de ressource ne peut être allouée à au plus une seule activité, on peut le voir comme un problème du flot, tel que chaque unité de ressource utilisée par une activité A_i doit être transférée à une seule autre activité A_j dès que l'activité A_i soit entièrement exécuté. Pour cela nous avons les variables suivantes :

$$x_{ij} = \begin{cases} 1 si\ l'activité \text{A}_i \text{ est exécutée avant } A_j \\ 0 sinon \end{cases}$$

$$f_{ij}^k = \text{quantité de ressources(flot) de } R_k \text{ allant de } A_i \text{ vers } A_j$$

Le modèle s'écrit alors comme suit :

$$
\begin{array}{lll}
 & (1) & min\ S_{n+1} \\
 & sc. & \\
 & (2) & \text{x}_{ij} = 1 \qquad \forall(\text{A}_i, A_j) \in \text{E} \\
 & (3) & \text{x}_{ij} + x_{ji} \leq 1 \qquad \forall(\text{i,j}) \in \text{V}^2,\ i < j \\
 & (4) & \text{x}_{ik} \geq x_{ij} + x_{jk} - 1 \qquad \forall(\text{i,j,k}) \in \text{V}^3 \\
(FF): & (5) & \text{S}_j \geq S_i - M + (p_i + M)x_{ij} \qquad \forall i, j \in V \\
 & (6) & \sum_{j \in A \cup \{A_{n+1}\}} f_{ij}^k = b_{ik} \qquad \forall i \in A \cup \{A_0\}, \forall k \in R \\
 & (7) & \sum_{i \in A \cup \{A_0\}} f_{ij}^k = b_{jk} \qquad \forall j \in A \cup \{A_{n+1}\}, \forall k \in R \\
 & (8) & 0 \leq f_{ij}^k \leq \min(b_{ik}, b_{jk}) x_{ij} \qquad \forall i, j \in V\ , \forall k \in R \\
 & (9) & x_{ij} \in \{0, 1\} \forall i, j \in V \\
 & (10) & \text{S}_i \geq 0 \forall i \in V
\end{array}
$$

L'objectif(1) est la minimisation de la date de complétion du projet, qui est la date de début de la tâche artificielle A_{n+1}. (2)modélisent les contraintes de précédences du problème. Les contraintes (3) et (4) garantissent respectivement la transitivité et l'absence de cycles dans le graphe de précédences. (5) lient les deux variables S et x : pour une valeur de M suffisamment large(section IV) cette contrainte impose que toute précédence $x_{ij} = 1$, l'activité A_j doit débuter après la complétion de la tâche A_i, autrement si $x_{ij} = 0$ la distance $S_j - S_i$ n'est pas contraint.
La contrainte (8) indique que si x_{ij} est nul, alors f_{ijk} aussi, et ce pour tout ressource R_k. Les contraintes (6) et (7) expriment le fait que le flot entrant et sortant d'une activité sur une ressource R_k est égal à la quantité d'unités de ressource R_k requise par cette tâche. Il faut noter que pour avoir une solution il faut initialiser b_{0k} et b_{n+1k} à B_k, et que M soit une constante qui vérifie : $S_i - S_j \leq M,\ \forall A_i, A_j \in V$ Cette formation contient un grand nombre de variables, un grand nombre de symétrie et de variables de flot, en plus la présence de la "Big-M" affaibli la relaxation continue du problème, mais un nombre polynomial de contraintes.

III.1.2 Configuration interdites

Cette formulation se base sur la notion de *ensemble critique minimal*[1] $C \in Cm$ de tâches. La formulation suivante est basée sur l'observation que dans toute solution réalisable, il faut que au moins une tâche de tout ensemble critique ne soit pas exécutée avec eux en même temps, c'est-à-dire : $\sum_{(i,j) \in C^2} (1 - x_{ij}) \leq |C| - 1$, autrement dit : $\sum_{(i,j) \in C^2} x_{ij} \geq 1$.
Le modèle est :

1. un ensemble de tâches dont la somme des consommations de ressources excède la quantité disponible d'au moins une ressource, et on dit qu'un ensemble critique est minimal si on supprimant n'importe quelle tâche de l'ensemble, on n'obtient pas un ensemble critique

$$
(CI):\quad
\begin{array}{ll}
(1) & min\ S_{n+1} \\
sc. & \\
(2) & \mathrm{x}_{ij} = 1 \qquad \forall(\mathrm{A}_i, A_j) \in \mathrm{E} \\
(3) & \mathrm{x}_{ij} + x_{ji} \leq 1 \qquad \forall(\mathrm{i,j}) \in \mathrm{V}^2,\ i < j \\
(4) & \mathrm{x}_{ik} \geq x_{ij} + x_{jk} - 1 \qquad \forall(\mathrm{i,j,k}) \in \mathrm{V}^3 \\
(5) & \mathrm{S}_j \geq S_i - M + (p_i + M)x_{ij} \qquad \forall i, j \in V \\
(6) & \sum_{(i,j)\in C^2} x_{ij} \geq 1 \qquad \forall C \in Cm \\
(7) & x_{ij} \in \{0,1\} \forall i, j \in V \\
(8) & \mathrm{S}_i \geq 0 \forall i \in V
\end{array}
$$

Au contraire de la formulation précédente, cette formulation contient un nombre exponentiel de contraintes. La génération de l'ensemble *Cm* prend trop de temps, pour cela dans ce travail nous ne générons qu'un ensemble *Cm'* d'ensembles critiques(section V), la solution qu'on obtient n'est donc optimale mais plutôt une bonne borne inférieure.

III.2 Modèles indexées sur le temps

Ces formulations dépendent fortement de l'horizon T qu'on suppose connu, en premier on la prend égale à la somme des durées des tâches, puis nous verrons comment l'améliorer dans la section IV, et on note $\tau = \{1, ..., T\}$ l'ensemble des instants de début possible aux tâches. Dans la formulation que nous présentons, nous ne n'utiliserons qu'une seule variable y_{jt} pour détecter l'instant t où la tâche A_j débute l'exécution.

$$
(IT):\quad
\begin{array}{ll}
(1) & min\ \sum_{t\in\tau} t.y_{n+1,t} \\
sc. & \\
(2) & \sum_{t\in\tau} y_{i,t} = 1 \qquad \forall \mathrm{i} \in \mathrm{V} \\
(3) & \sum_{t\in\tau} t.(y_{j,t} - y_{i,t}) \geq p_i \qquad \forall(\mathrm{i,j}) \in \mathrm{E} \\
(4) & \sum_{i\in V} b_{ik} \sum_{r=t-p_i+1}^{t} y_{ir} \leq B_k \qquad \forall k \in R, \forall t \in \tau \\
(5) & y_{it} \in \{0,1\} \forall i \in V, \forall t \in \tau
\end{array}
$$

l'objectif(1) minimisant le temps total d'exécution du projet. La première et la deuxième contraintes assurent la non-préemption et les contraintes de précédences des tâches, ainsi la $3^{ème}$ contrainte modélise les contraintes de ressources. l'avantage de cette formulation est qu'elle contient un nombre pseudo-polynomial de variables mais avec une mauvaise borne inférieure de relaxation continue.

IV. Pré-traitements des programmes linéaires et inégalités valides

Les techniques de prétraitement ont pour but d'éliminer des variables et de réduire le domaine des variables. Pour cela nous utiliserons des techniques de programmation par contraintes. Supposons qu'on connaît une fenêtre de temps pour chaque tâche ES_i, LS_i,

- dans les modèles en temps discrétisé, on pose $T = LS_{n+1}$
- on peut poser $y_{it} = 0\ \forall \mathrm{t} \notin ES_i, LS_i$ pour chaque tâche A_i
- On peut renforcé la valeur de la "Big-M" en la remplaçant pour chaque pairs de tâches i et j par $max\{LS_i, LS_j\} - min\{p_i, p_j\}$, avec cette valeur on a bien $S_i - S_j \leq M_{ij}$
- Soit i et j deux tâches en disjonction [2], alors i et j doivent nécessairement s'exécuter l'une après l'autre : $x_{ij} + x_{ji} = 1$.
  Ce concept peut se généraliser en un ensemble C de tâches deux à deux en disjonction, alors $x_{ij} + x_{ij} = 1 \forall (i,j) \in C^2$

2. il existe une ressource k de capacité inférieure strictement à la somme des quantités consommables par chacune des deux tâches, c'est-à-dire $b_{ik} + b_{jk} > B_k$

V. Construction des ensembles critiques et disjonctives

Comme l'énumération de tous les ensembles critiques est presque impossible, nous l'avant construire qu'un sous-ensembles, composé de 3 tâches, dont au plus 2 peuvent être exécutée en même temps.
On note $A_i||A_j$ si les deux tâches ne sont pas disjonctives.

Algorithm 2: tâches critiques

```
Result: liste : liste des triplet des tâches critiques
for i = 0, ..., n − 2 do
    for j = i + 1, ..., n − 1 do
        if A_i||A_j then
            for t = j+1 , ..., n do
                if A_i||A_j et A_i||A_t et A_j||A_t et A_i, A_j, A_t ne peuvent pas être ensemble then
                    liste ⟵ {i, j, t} end
                end
            end
        end
    end
```

Afin de trouver l'ensemble des tâches disjonctives nous avons appliquer l'algorithme suivant :

Algorithm 3: tâches critiques

```
Result: liste : liste des ensembles de tâches deux à deux en disjonction
for i = 0, ..., n − 2 do
    ensemble ⟵ {i}
    for j = i + 1, ..., n − 1 do
        temp ⟵ Vrai                                                    end
        for t ∈ ensemble do
            if A_i // A_t then
                temp ⟵ Faux
            end
        end
        if temp = Vrai then
            ensemble ⟵ {j}
        end
    end
    if |ensemble| > 1 then
        liste ⟵ {ensemble}
    end
```

VI. Borne inférieur

Il existe plusieurs techniques pour trouver une borne inférieure, la plus simple est appelée borne du chemin critique, qui s'obtient on résolvant le problème sans prendre en compte les contraintes de ressources. La résolution de ce problème s'effectue en temps polynomial, il suffit de trouver le plus long chemin entre le sommet de départ et tous les autres sommets.
Cependant la borne inférieure la plus utilisé est de la relaxation continue du problème, en relâchant les contraintes intégrités. Puis il existe des techniques spécifiques qui consistent à ajouter des inégalités valides à ce problème relâché après chaque résolution, tant que la solution trouvée

n'est pas entière.

Algorithm 4: Date au plus tôt des tâches

```
Data: Graph
Result: dist(s, t)
Notation :
dist(v) : distance entre s et v
N(v) : voisinage de v
c(v) : couleur de v, (tel que white : non visité, grey : découvert, black : visité)
initialisation : ∀v ∈ V(G)
c(v) = white
dist(v) = -∞
c(s) = grey    d(s) = 0
while ∃v:  c(v)=grey do
    v ⟵ max{dist(x):c(x)=grey}
    for w ∈N(v) do
        if c(u)=white then
            dist(w) ⟵ dist(v) + d(v, w)
            f(w) ⟵ v
            c(w) ⟵ grey
        end
        if c(w)=grey then
            if dist(w) > dist(v) + d(v, w) then
                dist(w) ⟵ dist(v) + d(v, w)
                f(w)
            end
        end
    end
    c(v) ⟵ black
end
```

On peut améliorer cette borne, on prenant en compte les ensembles crtiques, et les ensembles des tâches en disjonctions.

VII. Heuristiques

Le moyen le plus simple pour construire une solution réalisable au problème, est d'appliquer une méthode constructive, dite gloutonne, dont lequel on fixe itérativement une date de début à une tâche et on ne revient jamais à ce choix.
On premier on construit une liste d'ordre des tâches à traité compatible les contraintes de précédence. On obtient cette liste on considérant le problème sans contraintes de ressources, on se retrouve avec une problème d'ordonnancement avec précédence, dont la solution s'obtient en temps polynomial, il suffit de chercher le plus long chemin entre A_0 et tout les autres sommets.
Afin de rendre la solution réalisable, il existe deux principaux algorithmes de construction progressive *en série* et *en parallèle* :
L'algorithme en série ordonne les tâches suivant la liste, de telle sorte qu'à chaque itération la tâche sélectionnée s'exécute le plus tôt possible compte tenu des contraintes de ressources et de précédences.
L'algorithme en parallèle ordonnance les tâches selon l'ordre, tel qu'à chaque instant on exécute la première tâche disponible toute en respectant les contraintes de ressources et de précédences, si une telle tâche n'existe pas, on passe à l'instant suivante.
Chacune de ces heuristique permet d'obtenir une borne supérieure du problème(UB), et à partir cette borne supérieure on peut trouver les dates au plus tard (LS_i) de chaque tâche, qui est calculée comme le reste de la soustraction de $UB + p_i$ par le plus long chemin entre la tâche A_i et A_{n+1}.

VIII. Résultats

Nous avons implémenté les formulations et nous les avons testé sur l'ensemble de instances télécharger depuis le site http ://www.om-db.wi.tum.de/psplib/data.html.
Les programmes ont été écrits en langage C++, tous les tests ont été effectués sur un processeur Intel Core i5 1.70 et 6.00 Go RAM.
Les temps d'exécution en secondes et ont été limité à 1800 secondes(30 minutes).

La figure1 montre la différence entre les deux formulations(Flot et time indexed), et on remarque que la formulation basé sur les temps est plus performante que celle du flot.

Dans le tableau des résultats, on remarque que les valeurs de la relaxation linéaire des formulations sont très proches entre eux, elles ont presque la même qualité, mais la différence réside dans le temps de la recherche de la solution, tel qu'on remarque que la formulation de time indexed résolue le problème plus rapidement que la formulation de flot.

En ajoutant les inégalités valides de la section IV dans les formulations, nous avons remarqué que le temps de résolution diminue (Figure2).

Dans la Figure3, nous présentons la différence d'écart entre la solution obtenu par chaque algorithme et la solution heuristique, et d'après les résultats obtenus, on peut dire que la construction en parallèle est plus performante que la construction en série.

(Les résultats numériques sont dans le fichier *Resultats.xlsx*)

Références

[1] J.A. Hoogeveen-S.L. van de Veldeb-B. Veltmanc3, *"Complexity of scheduling multiprocessor tasks with precpecified processor allocations"*, Discrete Applied Mathematics 55 (1994) 259-272.

[2] J.BLAZEWICZ, J.K. LENSTRA, A.II.G. RINNOOY KAN, *" Scheduling subject to resource constraints : Classification and complexity"*, Discrete Applied Mathematics 5 (1983) 1 l-24

[3] Christian ARTIGUES *"The Resource-Constrained Project Scheduling Problem"*

FIGURE 1 – Comparaison des heuristiques

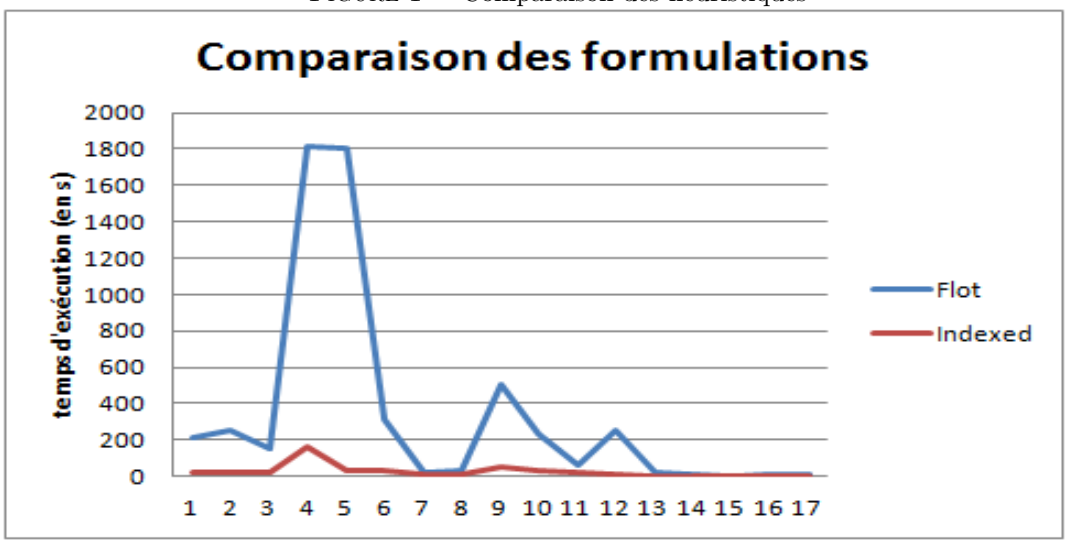

FIGURE 2 – Amélioration des temps d'exécution

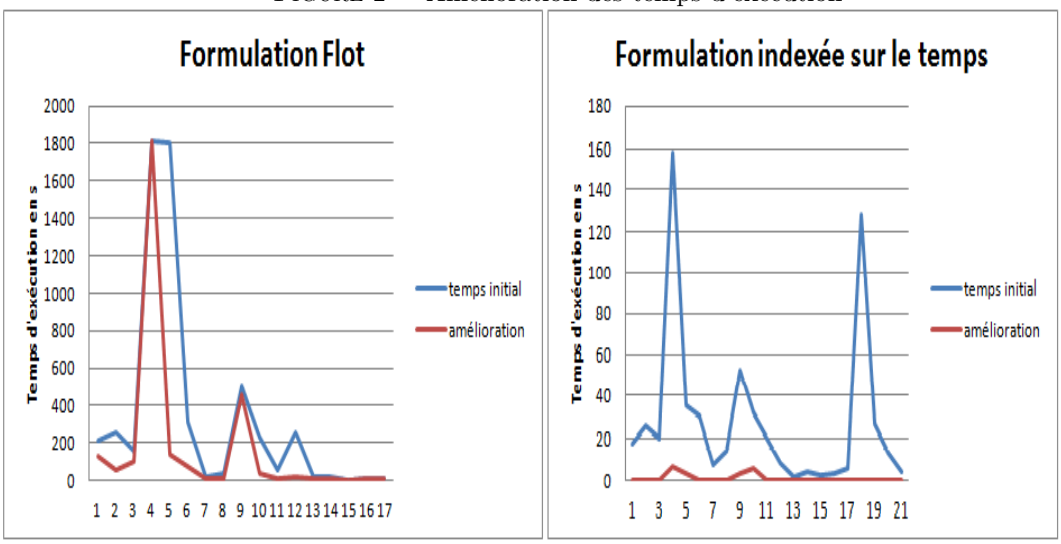

FIGURE 3 – Comparaison des heuristiques