Ordonnancement de Projet et contraintes de ressources

Vendredi 2 mai 2012

Larby KENNOUCHE Guillaume PERROTTON Satya TAMBY
Guillaume VANTROEYEN

Dauphine MIDO Décision

Table des matières

1 Introduction

L'objectif de ce rapport est de présenter différentes approches pour la résolution de problèmes d'ordonnancements de projets. Nous présenterons dans un premier temps le cas que nous étudierons, puis nous détaillerons notre méthode de résolution approchée du problème, basée sur un algorithme de listes. Enfin, nous concluerons sur un algorithme que nous avons imaginé (utilisant la méthode de séparation et évaluation) permettant d'obtenir une solution optimale.
Tous les algorithmes présentés dans ce rapport ont été implémentés en *Haskell*, le code source étant fourni en annexe.

2 Préliminaire

2.1 Ordonnancement sans contraintes de ressources

Dans cette partie, nous allons résoudre un problème d'ordonnancement de projet à 10 tâches et deux ressources ayant une disponibilité de 7 et 4. Pour cela, nous allons dans un premier temps tracer le PERT correspondant au problème avec ressources illimitées.

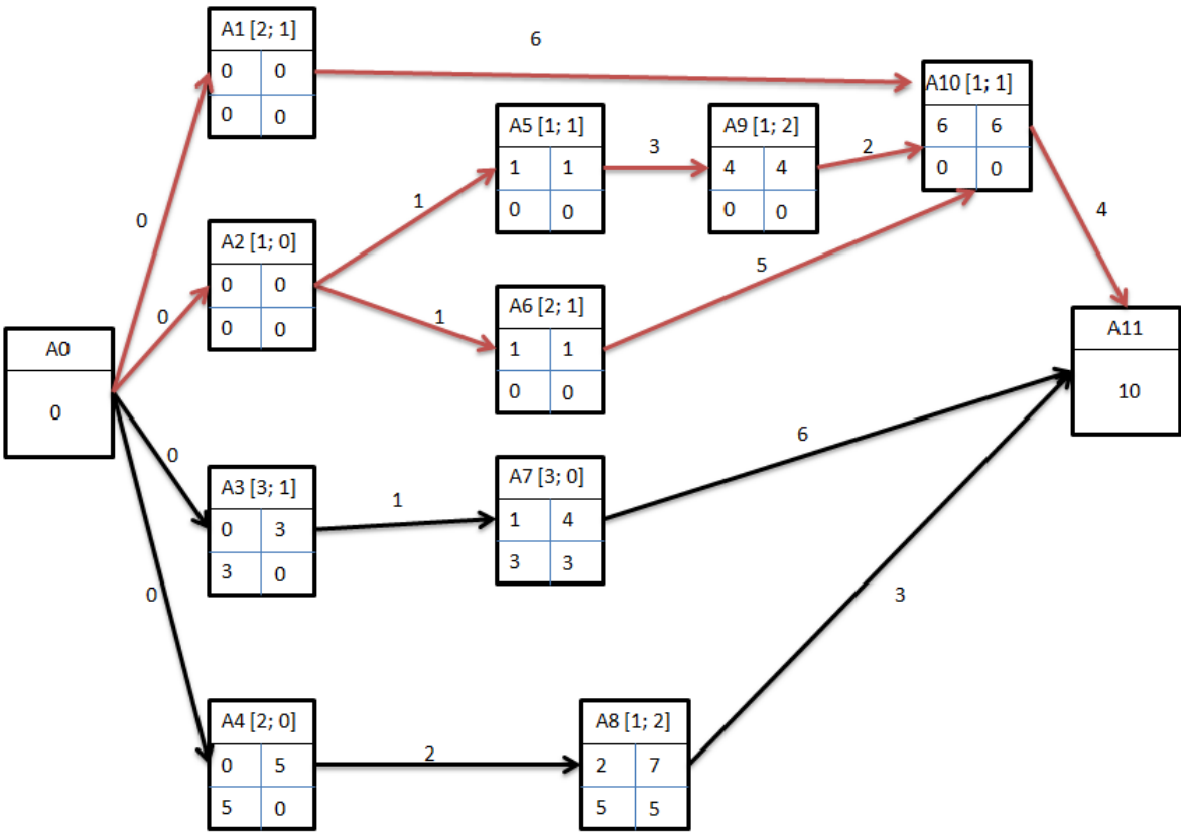

FIGURE 1 – PERT

Pour tracer ce graphe nous avons pris en compte que les contraintes de précédence. Nous avons calculé dans un premier temps les dates de début au plus tôt, puis celles de début au plus tard afin de déterminer successivement les marges libres et totales. Ainsi, nous avons mis en évidence le chemin critique en rouge. Et, on remarque que sans contraintes de ressource la date de fin au plus tôt est de 10 unités de temps.

Lorsque nous prenons en compte les contraintes de ressources, on observe que la solution trouvée précédemment ne peut être réalisé. En effet, on remarque qu'il n'est pas possible de démarrer les tâche $A1$, $A2$, $A3$, et $A4$ en même temps puiqu'elles nécessitent 8 unités de ressources 1 alors qu'il y en a que 7 disponibles.

Pour mettre en évidence ces violations de contraintes, nous avons tracé le diagramme de ressources correspondant au résultat optimal :

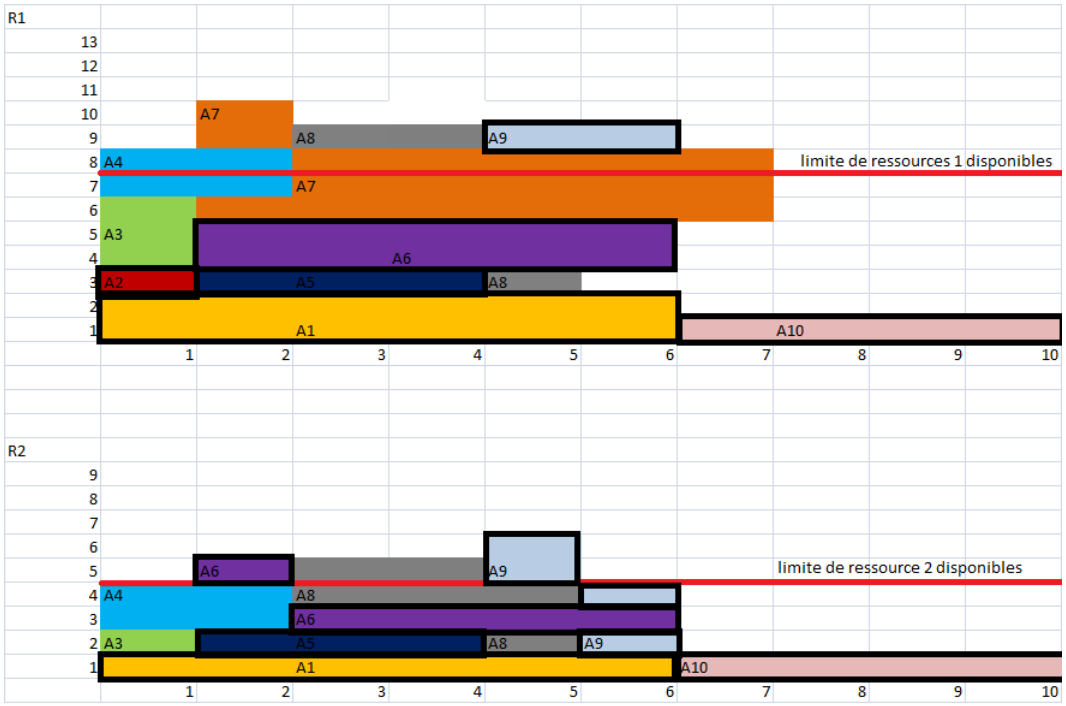

FIGURE 2 – Ordonancement optimal et ressources

Dans ce diagramme, nous avons entouré en noir les tâches critique. On sait qu'il est possible de démarer les tâches $A4$, $A8$, $A3$, et $A7$ plus tard (tâches non critiques). Nous allons donc analyser ce diagramme afin de tenter de trouver un ordonnancement au plus tôt qui respecte les contraintes liées aux ressources.
Cependant, on remarque que les contraintes liées à la ressource 1 ne peuvent être vérifiées. En effet, On observe que les tâches critiques utilisent 5 unités de ressources 1 entre les dates 4 et 6. Il y a donc 2 unités de ressources 1 disponibles entre les date 4 et 6. Or, d'aprés les dates de début au plus tôt et au plus tard de la tâche $A7$, celle-ci doit être en cours d'éxécution à ce moment. La tâche $A7$ utilise 3 unités de ressources 1, on en déduit que les ordonnancements de durée minimale ne peuvent pas satisfaire les contraintes de ressources. Pour satisfaire les ressources il faudrait décaler une des

tâches critiques et donc allonger la durée du projet.

2.2 Ordonnancement avec contraintes de ressources

Afin de résoudre le problème avec les contraintes de ressources, nous allons utiliser un algorithme de liste. Pour cela, lorsqu'il y a un conflit entre plusieurs tâches pour connaître laquelle doit débuter en premier, nous utilisons une heuristique nous permettant de faire ce choix. Les heuristiques utilisés dans l'ordre sont :

- Plus petite date de début au plus tard d'abord.
- Plus petite date de fin au plus tard d'abord.
- Plus petite durée d'abord.

Pour débuter l'algorithme,nous créons une liste contenant toutes les tâches pouvant débuter sans prendre en compte les ressources. Puis, nous prenons la meilleure tâche selon notre heuristique et la démarons. Par la suite, nous mettons à jour la liste des tâches pouvant débuter. Finalement, nous réitérons jusqu'à que cette liste soit vide.

Nous présentons dans un premier temps les différents états de la liste au fur et à mesure du déroulement de l'algorithme (les tâches en rouge sont celles selectionnées à chaque étape) , puis le GANTT obtenu finalement.

étape	1	2	3	4	5	6	7	8	9	10	11
liste des tâches disponible	A1 A2 A3 A4	A1 A3 A4 A5 A6	A3 A4 A5 A6	A3 A4 A6 A9	A3 A4 A9	A4 A7 A9	A4 A7 A10	A4 A10	A8 A10	A8	A11

FIGURE 3 – Déroulement de l'algorithme

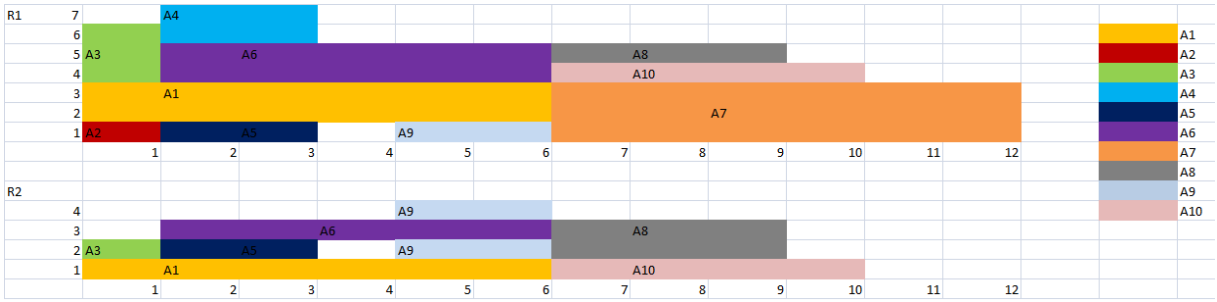

FIGURE 4 – GANTT

La solution trouvée est donc de 12 unités de temps : *A0(0 ;0) A1(0 ;6) A2(0 ;1) A3(0 ;1) A4(1 ;3) A5(1 ;3) A6(1 ;6) A7(6 ;12) A8(6 ;9) A9(4 ;6) A10(6 ;10) A11(12 ;12)*

3 Implémentation de l'algorithme

3.1 Présentation du problème général

Dans le cas général, le problème sans contraintes de ressources se présente sous la forme d'un graphe valué orienté représentant la dépendance des tâches. Ainsi, deux

sommets S_i et S_j sont reliés entre eux par un arc valué $v_{i,j}$ si la tâche j ne peut commencer que $v_{i,j}$ unités de temps après le début de la tâche i. Il est très important de remarquer, que dans la pratique, le graphe est **sans circuits de longueur strictement positive** car cela signifierait qu'une tâche dépendrait d'elle même, ce qui n'est en pratique pas le cas. L'objectif est de déterminer la date de début minimale de chaque tâche, c'est à dire **la valeur du plus long chemin entre le début du projet et chacune d'entre elles (date de début au plus tôt)**.

3.2 Implémentation de la méthode PERT

Pour résoudre le problème présenté ci-dessus il faut tout d'abord calculer les dates de débuts ainsi que les marges.En ce qui concerne la date de début au plus tôt, il suffit de remarquer que la date minimale du début d'une tâche est la plus petite date où la tâche est réalisable, c'est à dire où ses prédécesseurs sont satisfaites. Ainsi, il suffit de calculer l'ensemble des dates minimales de fin de chacune des prédécesseurs et d'en prendre la plus grande, car c'est au moins à partir de cet instant que la tâche que nous considérons peut commencer. Pour calculer la date minimale de fin d'un des successeurs, il suffit d'ajouter sa durée à sa date minimale de début.

$$\begin{cases} \lambda_{debut} = 0 \\ \lambda_j = max\left\{\lambda_i + v_{i,j} | i \leftarrow prdcesseurs(j)\right\} \end{cases}$$

De manière similaire, après avoir fixé la date de fin du projet nous pouvons alors déterminer les dates de débuts au plus tard par la formule :

$$\begin{cases} \lambda'_{fin} = D \\ \lambda'_i = min\left\{\lambda'_j - v_{i,j} | j \leftarrow successeurs(i)\right\} \end{cases}$$

En ce que concerne les marges totales, le calcul est trivial et pour les marges libres la formule est la suivante :

$$\begin{cases} ML_{fin} = 0 \\ ML_i = min\left\{\lambda_j - \lambda_i - v_{i,j} | j \leftarrow successeurs(i)\right\} \end{cases}$$

Il est important de remarquer que cet algorithme est sûr de terminer uniquement parce que le graphe est sans circuit de valeurs strictement positive. En effet, cela nous permet d'affirmer que nous passons une et une seule fois par chaque sommet et par chaque arc (en supposant qu'une fois que nous avons déterminé la valeur d'un optimum, nous la retenons pour le calcul des optima suivants).

Nous concluerons donc en disant que la méthode *PERT* - qui est, au final, une application de la programmation dynamique - permet de déterminer les dates de début au plus tôt en temps polynomial du nombre de sommets ajouté au nombre d'arrêtes.

3.3 Résolution du problème par contraintes de ressources

L'ordonnancement résultant de la méthode *PERT* est le meilleur que nous pouvons réaliser, en ne tenant compte que des dépendances des tâches. Ainsi, il est trivial que si nous rajoutons des contraintes au problème, la solution est généralement de durée plus longue. En effet, nous sommes également contraints de n'utiliser qu'un nombre

limité de ressources, et donc nous ne pouvons pas réaliser un nombre illimité de tâches simultanément.

Résoudre un tel problème est complexe car il faut considérer tous les conflits à chaque instant. C'est pourquoi la méthode que nous avons choisie initialement n'est pas optimale et ne se contente que de fournir un ordonnancement réalisable sous ces contraintes. Il est cependant possible d'améliorer la solution en choisissant une heuristique particulière pour la résolution des conflits de ressources, que nous pouvons déterminer selon le problème ou empiriquement.

Nous considérons qu'à chaque instant, nous possédons une liste des tâches qui sont *en cours de réalisation* et de celles qui sont *réalisables* (susceptibles d'être démarées car leurs dépendances sont réalisées). Voici le détail de ce que nous faisons à chaque itération (à chaque instant) :

- Nous déterminons, parmi les tâches en cours d'execution, celles qui sont terminées que nous retirons de la liste.
- Pour chaque tache *T* terminée, nous libérons les ressources qu'elle occupait, puis nous considérons tous ses successeurs T_i dont les dépendances sont réalisées, que nous ajoutons à la liste des tâches réalisables.
- Nous ordonnons la liste des tâches réalisables selon notre heuristique (e. g. par dates de début au plus tard croissantes).
- Nous la parcourons dans l'ordre, et, pour chaque tache réalisable T_r , si nous avons assez de ressources, nous la démarrons (on diminue la quantité totale de ressources, nous affectons à T_r une date de début, et nous l'ajoutons à la liste des tâches en cours d'execution).
- Nous avançons dans le temps jusqu'à ce qu'une tâche se termine (à la plus petite date de fin de chaque tâche en cours d'execution) et nous itérons jusqu'à ce qu'il n'y ait plus de tâches réalisables et plus de tâches en cours d'execution.

Il est trivial que l'algorithme terminera toujours s'il n'y a aucun circuit de durée strictement positive car il arrivera nécessairement un moment où toutes les tâches seront terminées. La complexité d'un tel algorithme est plus élevée, car à chaque itération il est nécessaire d'ordonner les tâches réalisables.

3.4 Résolution exacte par méthode d'évaluation et séparation

Nous avons songé à une méthode pour obtenir une solution optimale au problème de minimisation de la durée du projet sous contrainte de ressources, et nous avons constaté qu'il était très simple de l'implémenter à partir de notre algorithme précédent.

Il faut en pratique, considérer qu'un noeud est un **ordonnancement partiel** de p tâches parmi les n. Développer (*séparer*) le noeud revient à considérer l'ensemble des actions possibles, sachant que nous avons ordonnancé ces p tâches, c'est à dire :

- Si des tâches sont réalisables, et que nous avons les ressources nécessaires pour les réaliser, on peut choisir d'en executer une.
- S'il existe des tâches en cours d'execution, nous devons également considérer en plus la possibilité d'attendre que l'une d'entre elles se termine.

Pour évaluer l'interêt d'une de ces actions (*évaluation du noeud*), nous considérons que la durée optimale de l'ordonnancement resultant est forcément comprise entre deux valeurs :

– La date de fin au plus tôt du projet, donnée par la méthode *PERT*, sachant que certaines tâches ont des dates de début fixées. En effet, comme dit précédemment, la durée de la solution optimale du problème avec contraintes de ressources est nécessairement plus grande que celle de l'ordonnancement sans contraintes de ressources.
– La date de fin du projet, si nous lui appliquons l'algorithme de liste à partir de l'instant actuel. En effet, comme cet algorithme donne une solution approchée, la solution optimale est nécessairement de durée plus longue.

Ainsi, pour chacune de ces actions, nous l'effectuons et nous déterminons sa date de fin au plus tôt et celle de la solution par l'algorithme de listes. Si elles sont égales, c'est que nous ne pouvons pas faire mieux que l'algorithme de listes et qu'il est, par conséquent, inutile de développer le noeud (la solution trouvée suffit).

La solution donnée par cette méthode appliquée sur l'exemple du projet est identique à l'ordonnancement de liste trouvé initialement, ce qui nous permet donc d'affirmer qu'il est optimal (de durée $d = 12$). Cependant, il est clair que la complexité d'une telle résolution est très supperieure à celle de la méthode approchée : en effet, à chaque conflit, nous testons (au pire) toutes les actions possibles, et nous itérons pour chaque action, donc la résolution est exponentielle du nombre de conflits.

3.5 Présentation du programme

Dans l'optique de simplifier la résolution de tels problèmes d'ordonnancement de projet, nous avons fait le choix de réaliser un programme permettant, à partir d'un simple fichier texte contenant les différentes tâches et leurs dépendances mutuelles, de générer le graphe de potentiels-tâches (*PERT*), de déterminer les dates au plus tôt, au plus tard ainsi que les différentes marges pour chacune, puis enfin de déterminer un ordonnancement réalisable pour le même problème sous contraintes de ressources limitées.

Afin de pouvoir vérifier les résultats, il est également possible de générer le programme linéaire de calcul des ordonnancements optimaux au format *CPLEX* (.lp) pour pouvoir le résoudre ulterieurement. Afin de vérifier notre programme nous avons dans un premier temps vérifier son comportement sur le graphe d'exemple du cours, les résultats sont présentés en *Annexes 1*. Nous nous interressons ici à l'exemple du projet.

Nous transcrivons le graphe dans un fichier texte qui va être lu par le programme :

```
1  ressources= [7,4]
2  nom=    A0  A1  A2  A3  A4  A5  A6  A7  A8  A9  A10 A11
3  id=     0   1   2   3   4   5   6   7   8   9   10  11
4  cout=   [0,0]   [2,1]   [1,0]   [3,1]   [2,0]   [1,1]   [2,1]   [3,0]   [1,2]   [1,2]   [1,1]   [0,0]
5  duree=  0   6   1   1   2   3   5   6   3   2   4   0
6  prec=   []  [0] [0] [0] [0] [2] [2] [3] [4] [5] [1,6,9] [0,10,8,9,7]
```

FIGURE 5 – Fichier de création du graphe

Ce dernier calcule alors les dates de début au plus tôt, au plus tard, les marges libres et totales, puis en appliquant l'algorithme de liste, un ordonancement au plus tôt (optimal ou proche de l'optimal) vérifiant les contraintes de coût. Ces données sont écrites un fichier :

```
Ordonnancement final :
[t=0.0] A0  id=0    duree=0.0   +tot=0.0    +tard=0.0   mtotale=0.0 mlibre=0.0  prec=[] nxt=[11,4,3,2,1]
[t=0.0] A1  id=1    duree=6.0   +tot=0.0    +tard=0.0   mtotale=0.0 mlibre=0.0  prec=[0]    nxt=[10]
[t=0.0] A2  id=2    duree=1.0   +tot=0.0    +tard=0.0   mtotale=0.0 mlibre=0.0  prec=[0]    nxt=[6,5]
[t=0.0] A3  id=3    duree=1.0   +tot=0.0    +tard=3.0   mtotale=3.0 mlibre=0.0  prec=[0]    nxt=[7]
[t=1.0] A4  id=4    duree=2.0   +tot=0.0    +tard=5.0   mtotale=5.0 mlibre=0.0  prec=[0]    nxt=[8]
[t=1.0] A5  id=5    duree=3.0   +tot=1.0    +tard=1.0   mtotale=0.0 mlibre=0.0  prec=[2]    nxt=[9]
[t=1.0] A6  id=6    duree=5.0   +tot=1.0    +tard=1.0   mtotale=0.0 mlibre=0.0  prec=[2]    nxt=[10]
[t=4.0] A9  id=9    duree=2.0   +tot=4.0    +tard=4.0   mtotale=0.0 mlibre=0.0  prec=[5]    nxt=[11,10]
[t=6.0] A7  id=7    duree=6.0   +tot=1.0    +tard=4.0   mtotale=3.0 mlibre=3.0  prec=[3]    nxt=[11]
[t=6.0] A8  id=8    duree=3.0   +tot=2.0    +tard=7.0   mtotale=5.0 mlibre=5.0  prec=[4]    nxt=[11]
[t=6.0] A10 id=10   duree=4.0   +tot=6.0    +tard=6.0   mtotale=0.0 mlibre=0.0  prec=[1,6,9]    nxt=[11]
[t=12.0] A11    id=11   duree=0.0   +tot=10.0   +tard=10.0  mtotale=0.0 mlibre=0.0  prec=[0,10,8,9,7]   nxt=[]
```

FIGURE 6 – Graphe Complet

Interpretation : La première colonne correspond à la date de début des tâches dans l'ordonancement vérifiant les contraintes de ressource, par exemple la tâche *A9* commence au temps 4. Viens ensuite les colonnes *nom* : nom de la tâche, *id* : relatif au programme, *+tot* : date de début au plus tôt, *+tard* : date de début au plus tard, *mtotale* : marge totale, *mlibre* : marge libre, *prec* et *nxt* les predecesseurs et successeurs.

```
Utilisation des ressources :
R1        R2      Instant Détail

******    |**     |t=0    ||A1 (27%), A2 (9%), A3 (36%)
*******   |***    |t=1    ||A1 (27%), A4 (18%), A5 (18%), A6 (27%)
*******   |***    |t=2    ||A1 (27%), A4 (18%), A5 (18%), A6 (27%)
*****     |***    |t=3    ||A1 (27%), A5 (18%), A6 (27%)
*****     |****   |t=4    ||A1 (27%), A6 (27%), A9 (27%)
*****     |****   |t=5    ||A1 (27%), A6 (27%), A9 (27%)
*****     |***    |t=6    ||A7 (27%), A8 (27%), A10 (18%)
*****     |***    |t=7    ||A7 (27%), A8 (27%), A10 (18%)
*****     |***    |t=8    ||A7 (27%), A8 (27%), A10 (18%)
****      |*      |t=9    ||A7 (27%), A10 (18%)
***       |       |t=10   ||A7 (27%)
***       |       |t=11   ||A7 (27%)
          |       |t=12   ||
```

FIGURE 7 – Utilisation des ressources

Interpretation : La suite du fichier nous indique l'utilisation de chaque ressources à chaque instant, une étoile par unité consommé, ansi que la répartition des ressource en pourcentage par tâche.

4 Conclusion

Nous avons apprécié de pouvoir utiliser nos connaissances acquises lors du premier semestre en optimisation combinatoire dans un domaine appliqué à l'ordonnancement de projet. Faire le choix de rédiger nos algorithmes en *Haskell* (langage fonctionnel assez récent rappelant le OCaml) s'est révélé judicieux car sa syntaxe permet de s'exprimer de manière aussi simple et rigoureuse qu'en mathématiques.

5 Annexes

5.1 Annexe 1 : Exemple du cours

Voici le graphe du cours transcrit dans un fichier, ansi que le fichier résultat du programme :

```
ressources= [5,1]
nom=            debut   A   B   C   D   E   F   G   H   I   J   B_prec Fin
id=     0   1   2   3   4   5   6   7   8   9   10  11  12
cout=       [0,0]   [3,0]   [3,0]   [1,0]   [1,1]   [1,1]   [2,1]   [3,0]   [2,1]   [1,0]   [0,0]   [0,0]   [0,0]
duree=      0   10  4   2   8   6   5   9   2   7   4   2   0
prec=       []  [0] [11]    [0] [1,2]   [2] [3] [4,5]   [5,6]   [7,8]   [6] [0] [9,10]
```

FIGURE 8 – Graphe initial

```
Ordonnancement final :
[t=0.0] debut   id=0    duree=0.0   +tot=0.0    +tard=0.0   mtotale=0.0 mlibre=0.0  prec=[] nxt=[11,3,1]
[t=0.0] A   id=1    duree=10.0  +tot=0.0    +tard=0.0   mtotale=0.0 mlibre=0.0  prec=[0]    nxt=[4]
[t=0.0] C   id=3    duree=2.0   +tot=0.0    +tard=18.0  mtotale=18.0    mlibre=0.0  prec=[0]    nxt=[6]
[t=0.0] B_prec  id=11   duree=2.0   +tot=0.0    +tard=4.0   mtotale=4.0 mlibre=0.0  prec=[0]    nxt=[2]
[t=2.0] F   id=6    duree=5.0   +tot=2.0    +tard=20.0  mtotale=18.0    mlibre=0.0  prec=[3]    nxt=[10,8]
[t=7.0] J   id=10   duree=4.0   +tot=7.0    +tard=30.0  mtotale=23.0    mlibre=23.0 prec=[6]    nxt=[12]
[t=10.0] B  id=2    duree=4.0   +tot=2.0    +tard=6.0   mtotale=4.0 mlibre=0.0  prec=[11]   nxt=[5,4]
[t=14.0] D  id=4    duree=8.0   +tot=10.0   +tard=10.0  mtotale=0.0 mlibre=0.0  prec=[1,2]  nxt=[7]
[t=22.0] E  id=5    duree=6.0   +tot=6.0    +tard=12.0  mtotale=6.0 mlibre=0.0  prec=[2]    nxt=[8,7]
[t=28.0] G  id=7    duree=9.0   +tot=18.0   +tard=18.0  mtotale=0.0 mlibre=0.0  prec=[4,5]  nxt=[9]
[t=28.0] H  id=8    duree=2.0   +tot=12.0   +tard=25.0  mtotale=13.0    mlibre=13.0 prec=[5,6]  nxt=[9]
[t=37.0] I  id=9    duree=7.0   +tot=27.0   +tard=27.0  mtotale=0.0 mlibre=0.0  prec=[7,8]  nxt=[12]
[t=44.0] Fin    id=12   duree=0.0   +tot=34.0   +tard=34.0  mtotale=0.0 mlibre=0.0  prec=[9,10] nxt=[]
```

FIGURE 9 – Résultats

Interpretation : Nous pouvons constater que l'ordonancement est bien le même que celui du cours après l'application manuelle de l'algorithme de liste. Voici l'utilisation des ressources pour chaque instant :

```
Utilisation des ressources :
R1      R2          Instant         Détail

****            |           |t=0          ||A (50%), C (16%), B_prec (0%)
****            |           |t=1          ||A (50%), C (16%), B_prec (0%)
*****           |*          |t=2          ||A (50%), F (50%)
*****           |*          |t=3          ||A (50%), F (50%)
*****           |*          |t=4          ||A (50%), F (50%)
*****           |*          |t=5          ||A (50%), F (50%)
*****           |*          |t=6          ||A (50%), F (50%)
***             |           |t=7          ||A (50%), J (0%)
***             |           |t=8          ||A (50%), J (0%)
***             |           |t=9          ||A (50%), J (0%)
***             |           |t=10          ||J (0%), B (50%)
***             |           |t=11          ||B (50%)
***             |           |t=12          ||B (50%)
***             |           |t=13          ||B (50%)
*               |*          |t=14          ||D (33%)
*               |*          |t=15          ||D (33%)
*               |*          |t=16          ||D (33%)
*               |*          |t=17          ||D (33%)
*               |*          |t=18          ||D (33%)
*               |*          |t=19          ||D (33%)
*               |*          |t=20          ||D (33%)
*               |*          |t=21          ||D (33%)
*               |*          |t=22          ||E (33%)
*               |*          |t=23          ||E (33%)
```

```
*              |*          |t=24         ||E (33%)
*              |*          |t=25         ||E (33%)
*              |*          |t=26         ||E (33%)
*              |*          |t=27         ||E (33%)
*****          |*          |t=28         ||G (50%), H (50%)
*****          |*          |t=29         ||G (50%), H (50%)
***            |           |t=30         ||G (50%)
***            |           |t=31         ||G (50%)
***            |           |t=32         ||G (50%)
***            |           |t=33         ||G (50%)
***            |           |t=34         ||G (50%)
***            |           |t=35         ||G (50%)
***            |           |t=36         ||G (50%)
*              |           |t=37         ||I (16%)
*              |           |t=38         ||I (16%)
*              |           |t=39         ||I (16%)
*              |           |t=40         ||I (16%)
*              |           |t=41         ||I (16%)
*              |           |t=42         ||I (16%)
*              |           |t=43         ||I (16%)
               |           |t=44         ||
```

Cependant, cet exemple constitue également l'un des cas où l'algorithme de listes ne donne pas une solution optimale. Ainsi, notre algorithme d'évaluation et séparation nous permet d'aboutir à une solution de durée plus courte (40 jours au lieu de 44). Il est interessant de constater que l'algorithme choisit d'executer uniquement la tâche C, alors qu'il reste suffisament de ressources pour executer d'autres tâches réalisables. Voici le diagramme de Gantt que nous avons tracé à partir des résultats :

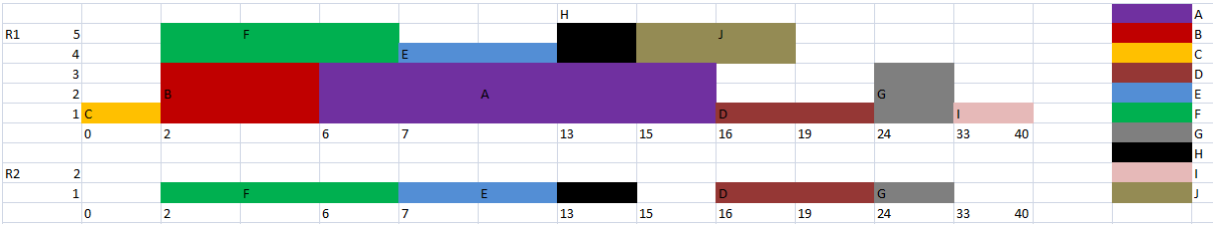

FIGURE 10 – GANTT