Optimisation multi-objectif pour l 'ordonnancement

Bentellis Amine
Université Nice Sophia-Antipolis

16 Juin, 2020

Résumé

Ce travail se concentre sur les problèmes d'ordonnancement ainsi que l'optimisation multiobjectif. Dans un premier temps, on présente les réseaux de contraintes temporelles du type PERT/CPM et leur implémentation en programmation linéaire. Ensuite, on apporte une définition du préordre leximin suivi d'une comparaison avec Min-max fairness. Enfin, on applique et utilise ce qui a été présenté sur un exemple de problème PERT.

1 Introduction

La réalisation d'un projet dans les délais n'est pas une tâche facile. Plusieurs facteurs peuvent retarder un projet causant des coûts supplémentaires pour toutes les parties concernées. En effet un projet peut être découpé en plusieurs tâches, avec des durées de complétion différentes, chacune de ces tâches peut nécessiter des ressources matérielles ou humaines. Ce type de projet peut varier de la construction de maisons jusqu'à la création d'un software. Les outils les plus prédominants qui ont été prouvé et testés sont le Critical path Method (CPM) et le Project Evaluation and Review Technique (PERT). Grâce à ces méthodes développées dans les années 60 par l'US Navy, il est possible de déterminer les dates de début et de fin de projet. En particulier, la méthode CPM permet d'établir un ensemble de tâches dites critiques. Ces dernières sont cruciales pour une bonne gestion de projets car un retard sur l'une d'elles engendrerait un retard sur tout le projet. Contrairement aux autres tâches qui possèdent donc une marge de manoeuvre et dont le retard n'entrainerait pas de grosse répercussions sur la totalité du projet. Couplée à ce problème PERT/CPM s'ajoute la répartition juste de ressources dans un système. Ce problème présenté ici est directement lié au sens du terme équité. L'objectif de ce projet est d'étudier des problèmes d'optimisation multiobjectif sur un réseau de contraintes temporelles du type PERT/CPM. Pour cela nous utiliserons des techniques d'optimisation qui maximisent la juste répartition des biens. En effet, Max-min fairness ou encore leximin font partie des méthodes présentées dans ce rapport. Dans un

premier temps, il faut présenter le problème PERT puis comment optimiser les résultats en utilisant les méthodes décrites plus haut.

2 Présentation du problème PERT/CPM

Le problème PERT est très étudié et documenté cependant il est essentiel de le rappeler pour comprendre l'intégralité de ce travail. Tout d'abord PERT est une technique en gestion de projets qui permet de décrire et d'analyser les tâches d'un projet. Grâce à cette méthode il est possible de suivre de manière logique le réseau de travaux à réaliser. Le problème est représenté par un graphe de tâches et ainsi dégagé un planning précis des tâches à effectuer. Pour chaque tâche ou noeud, sont indiquées une date de début et de fin au plus tôt et au plus tard. Par ailleurs du diagramme il est possible de déterminer le chemin critique et spécifier la durée minimale du projet. Le but est d'obtenir une ordonnance des tâches optimales pour minimiser la durée du projet.
Soit un graphe G de tâches. Chaque tâche t contient :

- ***d*** la durée estimée de la tâche
- ***ES*** date de début au plus tôt
- ***EF*** date de fin au plus tôt
- ***LS*** date de début au plus tard
- ***LF*** date de fin au plus tard
- ***slack*** la marge qui correspond à LF - EF

Pour avoir une solution optimale qui satisfait le problème il faut commencer par créer le diagramme. Il y a deux types de diagrammes soit AOA (activity on arrow, en francais : activités sur les arcs) ou AON(activity on node, en francais : activités sur les noeuds). AON est plus simple à comprendre et implémenter. En programmation linéaire, on peut attribuer à chaque activité une variable $v_i,\ i \in [1, n]$. Puis la fonction objectif est :

$$minimize\ EF(v_n) - EF(v_1)$$

Ici on minimise la durée du projet car cette soustraction correspond à la durée totale du projet (v1 première tâche, vn dernière tâche). Cependant il faut être sur que les contraintes sont bien définis. Pour chaque tâches v_j précédé par une tâche v_i il faut que :

$$ES[v_j] - ES[v_i] \ \geq \ d(v_i)$$

Et : $EF = ES + d$

Chaque tâches v_j doit se produire après chaque tâches précédente v_i par au moins la durée de v_i . Mais aussi on précise que la date de fin au plus tôt correspond simplement à la date de début plus sa durée d. En implémentant ces deux contraintes on a un diagramme PERT avec les dates au plus tôt de fin de projet. Cette étape s'appelle le forward pass et est essentiel pour calculer les deux autres variables de décisions LS et LF. La seconde étape s'appelle le backward pass. En programmation linéaire, il suffit d'ajouter les

contraintes suivantes :

$LS = LF - d$

Et : $LF(vi) = min(LS(n),\ n \in Adj$ liste d'adjacence de $v_i)$

La première contrainte permet d'assurer que la date de début au plus tard correspond simplement à la date de fin au plus tard moins sa durée d. Et enfin la dernière contrainte sélectionne le minimum de LS parmi les noeuds adjacents pour déterminer la date de fin au plus tard.

On peut en déduire la marge que chaque tâche possède pour être fini en soustrayant la date de fin au plus tard avec celle au plus tôt.

$slack = LF - EF$

Finalement, la dernière étape est de déterminer le chemin critique autrement dit CPM. Avec le diagramme PERT fait cette étape est très simple. Il suffit de choisir les noeuds avec un slack égal à zéro. CPM détermine le plus long chemin du début vers la fin du projet. Ce processus détermine quelles activités sont "critiques" (c'est-à-dire sur le chemin le plus long) et lesquelles ont une "marge positive" (c'est-à-dire qu'elles peuvent être retardées sans changer la durée du projet). Le fait de retarder une des tâche sur le chemin critique entraine forcément un retard sur le projet entier.

3 Optimisation Leximin

L'optimisation leximin est un concept basé sur le partage équitable des ressources. L'équité est un terme subjectif mais important dans un large ensemble de problèmes du monde réel impliquant des ressources de travail. C'est aussi un concept utilisé pour allouer équitablement la bande passante sur un réseau ou encore le partage de l'espace aérien entre différentes compagnies aériennes. Ces problèmes cités plus haut sont documentés et plusieurs implémentations ont déjà été réalisées. Outre le fait que leximin formalise le terme équité dans des contextes impliquant plusieurs agents, il est également pertinent dans d'autres contextes, tels que la rupture de symétrie dans certains problèmes de programmation par contraintes (symmetry-breaking constraints).Commençons par définir formellement la notion de préordre leximin.

Définition Préordre Leximin : Soit deux vecteurs $\vec{x}$ et $\vec{y} \in \mathbf{N}^n$. $\vec{x}$ et $\vec{y}$ sont dits leximin-indifférents si et seulement si $\vec{x}^{\uparrow} = \vec{y}^{\uparrow}$. $\vec{y}$ est préféré strictement à $\vec{x}$ (noté $\vec{x} \prec_{leximin} \vec{y}$) si et seulement si $\exists k \in [0, n-1]$ tel que $\forall i \in [1, k]$, $x \uparrow_i = y \uparrow_i$ et $x \uparrow_{k+1} < y \uparrow_{k+1}$

Par exemple, si $\vec{x} =< 3, 2, 1, 2 >$ et $\vec{y} =< 5, 1, 1, 3 >$ alors on a $\vec{y} \prec_{leximin} \vec{x}$.

On se rend compte donc que le préordre leximin est un outil puissant pour attribuer équitablement une ressource. Par ailleurs, plusieurs articles présentent des méthodes et algorithmes utilisant leximin pour résoudre certains problèmes. En effet, il est possible d'utiliser le *framework* que la pro-

grammation par contraintes apporte pour résoudre des problèmes d'ordonnancement, planning, et allocation de ressources. Nous verrons par la suite que ma méthode n'utilise pas la programmation par contraintes mais s'inspire de certains algorithmes présentés dans [1]. Il serait intéressant aussi de noter que d'autres méthode que leximin existe tel que Max-min fairness. Effectivement,[5] explique que les définitions de min-max fairness, vecteurs lexicographiquement minimaux et vecteurs leximax minimaux sont analogues. Les mêmes relations existent entre Min-max fairness et la minimisation leximax.

Dans notre cas, l'optimisation multicritère a très peu été étudié dans ce type de réseau temporel. On a vu tout au long de cette partie que leximin permettait d'attribuer équitablement des ressources. Ici la ressource la plus importante est le temps. Donc l'idée de leximin dans un réseau PERT est d'attribuer équitablement du temps aux équipes ou tâches qui en ont le plus besoin. Cependant on ne peut pas attribuer du temps à n'importe quelle tâche car certaines se trouvent sur le chemin critique. Comme expliquer dans la partie 2, si on accorde du temps à ces tâches cela entrainerait un inévitable retard sur la date de fin du projet. Donc on se contentera ici d'accorder du temps qu'aux tâches avec une marge supérieure à zéro ($slack > 0$). Prenons l'exemple du **problème des flux concurrents maximaux** (*maximum concurrent flow problem* en anglais) qui est défini comme suit :

Soit un réseau N avec multiples flux de marchandises et un ensemble de marchandises D. Pour chaque demande d, l'objectif est d'envoyer T_d unités de marchandises de la source s^{d} à sa destination t^{d}.

Pour cet exemple, appliqué leximin est plutôt simple car on a une demande d précise et un nombre de marchandises défini. Donc on a une borne inférieure ainsi qu'une borne supérieure pour chaque flux. Pour revenir à notre problème, la seule information qu'on possède sur le temps t que l'on peut ajouter à une tâche est $t \in [0, slack]$. Il faut donc ajouter des contraintes relatives à t pour limiter le temps qu'on peut accorder au total.

4 Présentation du Modèle

4.1 Construction du réseau PERT

Le modèle présenté ici utilise dans un premier temps la programmation linéaire pour dégager un graphe PERT des tâches en entrées. Une fois que le vecteur slack a été calculé le modèle calcule la solution leximin maximale.

Dans un premier temps, en utilisant la définition de PERT plus haut, on peut écrire un programme linéaire avec les mêmes contraintes sur les dates de fin et début du projet. Tout d'abord, pour résoudre le problème d'ordonnancement il faut établir quelles sont les durées de chaque tâche ainsi que la liste de ses prédécesseurs. L'exemple qui est utilisé est celui d'un

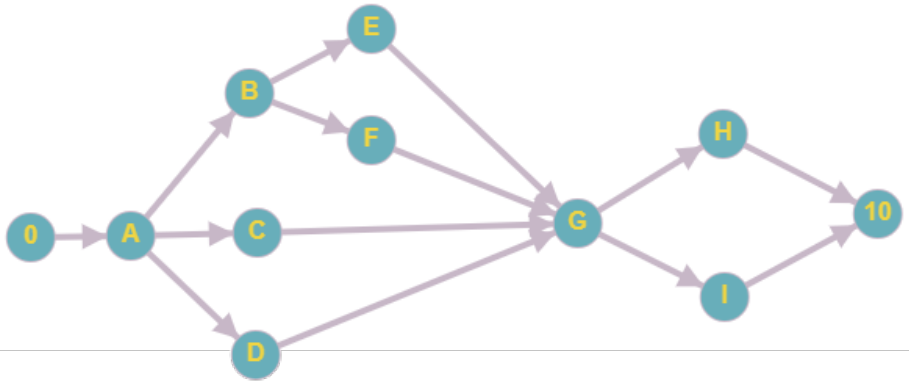
projet de construction d'une maison, des fondations jusqu'à l'aménagement intérieur et extérieur.

TABLE 1 – Description d'un projet de construction d'une maison

Tâches	Description	Prédécesseurs	Durée Estimée	Équipe
0	Début de Projet	-	0	0
A	Fondations	0	7	1
B	Murs	A	7	2
C	Plancher	A	6	2
D	Toiture	A	3	3
E	Peinture	B	5	1
F	Travaux Électriques	B	3	0
G	Fenêtres & Portes	C,D,E,F	4	3
H	Aménagement Intérieur	G	4	0
I	Aménagement Extérieur	G	3	1
10	Fin du Projet	H,I	0	0

Les informations apportées dans ce planning sont similaires à ce qu'on peut entrer dans un programme de gestion de projets comme MS Projects. Dans cet exemple, on considère que quatre équipes différentes se partagent le projet. On peut ainsi créer le graphe correspondant au planning des tâches du projet.

FIGURE 1 – Graphe représentant le projet de construction d'une maison



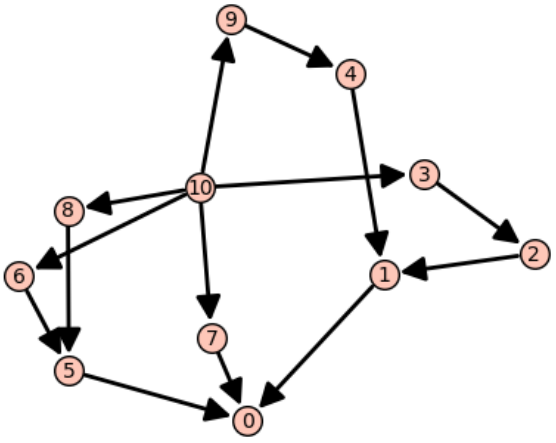
Maintenant, si on applique les contraintes énoncées dans la partie 2 on retrouve la totalité des informations pertinentes pour un réseau PERT/CPM.

TABLE 2 – Date de début et de fin au plus tôt et au plus tard, slack et si la tâche se trouve sur le chemin critique

Tâches	EF	ES	LF	LS	Slack	Chemin Critique
0	0	0	0	0	0	OUI
A	7	0	7	0	0	OUI
B	14	7	14	7	0	OUI
C	13	7	19	13	6	NON
D	10	7	19	16	9	NON
E	19	14	19	14	0	OUI
F	17	14	19	16	2	NON
G	23	19	23	19	0	OUI
H	27	23	27	23	0	OUI
I	26	23	27	24	1	NON
10	27	27	27	27	0	OUI

Par ailleurs, plusieurs autres tests ont été faits sur des configurations variables. Les graphes ont été générés grâce à *SageMath* pour python. Les *RandomDirectedGN* assurent que le graphe soit orienté, acyclique et avec un puits t unique. Pour générer un graphe PERT il suffit donc d'ajouter une source s avec des arcs vers toutes les sources s_i du graphe. La durée d de chaque tâche est déterminé aléatoirement entre 1 et 10. De la même manière le nombre d'équipes et l'attribution des tâches sont faits aléatoirement.

FIGURE 2 – Exemple de *RandomDirectedGN* générer par Sage avec 11 sommets



Ainsi, trois configurations on été créées :
— ***ExampleGraph.dat*** : 11 sommets, 4 équipes.
— ***MediumGraph.dat*** : 30 sommets, 5 équipes.

— ***LargeGraph.dat*** : 500 sommets, 20 équipes.

TABLE 3 – Statistiques de chaque configurations

	ExampleGraph	MediumGraph	LargeGraph
Temps d'exécution(secondes)	0	0	4,35
Variables	100	439	107861
Contraintes	113	466	108504
Solution(Temps minimal du projet)	27	29	43

4.2 Optimisation leximin pour PERT

À présent, le réseau est établi et on connaît la marge que chaque tâche possède. On peut appliquer ce qu'on a vu en partie 3 pour déterminer une solution leximin optimale. La solution se présente sous la forme d'un vecteur $\vec{\lambda}$, $\forall i \in [0, n]$, on a $\lambda_i \in [0, slack_i]$ où n représente le nombre de tâches dans le réseau et $slack_i$ la valeur du $slack$ pour la tâche i.La solution leximin optimale est donc le vecteur $\vec{\lambda}$ si pour chaque solution non-optimale $\vec{x}$, on a $\vec{x} \prec_{leximin} \vec{\lambda}$. Dans notre cas, on explique plus haut qu'on a besoin d'autres contraintes car les tâches du réseau ne possèdent pas de demandes prédéfinies. Sans implémenter de contraintes on a :

$\vec{\lambda} = \vec{slack} =< 0, 0, 0, 6, 9, 0, 2, 0, 0, 1, 0 >$

On pourrait imaginer plusieurs façon d'implémenter ces contraintes. La première serait d'avoir un temps maximum t_{max} qu'on peut accorder en plus à l'ensemble d'un projet. Par exemple, pour $t_{max} = 7$ dans l'exemple plus haut on a :

$\vec{\lambda} =< 0, 0, 0, 1, 4, 0, 2, 0, 0, 0, 0 >$

Néanmoins, cette contrainte ne semble pas être la bonne façon d'appréhender ce problème. La deuxième possibilité serait d'être équitable dans le temps accorder à chaque les équipes. C'est à dire que $t_0 = t_1 = ... = t_m$, où t_i correspond au temps accorder à une équipe i et m le nombre d'équipes. Par ailleurs, on présume que le travaille a été réparti équitablement entre les équipes. En appliquant cette méthode on a :

$\vec{\lambda} =< 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0 >$

Il faut donc appliquer un ordre leximin sur l'ensemble des solutions pour trouver la solution leximin optimale. Par conséquent,la méthode la plus logique à implémenter est de comparer chaque solutions qu'on obtient par rapport à l'ordre leximin et donc de déterminer une solution $\vec{\lambda}$ qui sera optimale. C'est la méthode qui a été implémenter pour ce rapport en *CPLEX* grâce au langage *IBM ILOG Script* pour OPL. En effet, dans un premier temps on créer une configuration $rc1$ qui calcule et output le vecteur $\vec{slack}$ dans un fichier *.dat*. Puis, une configuration $rc2$ utilise ce fichier *dat* pour

calculer les solutions et ainsi déterminer la solution leximin optimale.

Cependant, cette méthode n'est pas la meilleure car certains articles ([1],[3]) montrent qu'il est possible d'implémenter leximin grâce à la programmation par contraintes et ainsi résoudre le problème en temps polynomial. Une variante de la méthode utilisé dans ce rapport qui est présenté dans [1] serait de définir un réseau de contraintes $R = (X, D, C)\ ; < u_1, ..., u_n >$ $i \in X^{\mathrm{n}}$. La première étape est de calculer une première solution, puis tenter de l'améliorer en précisant que la la solution suivante doit être meilleure (*leximin-preferred*) à la solution actuelle avec une contrainte ***Leximin***, et ainsi de suite jusqu'à ce qu'il n'y ait plus de solution à R. Effectivement, utiliser CP pour ce problème s'avère être la meilleure implémentation.

5 Conclusions & Discussion

Pour conclure, l'optimisation multicritère en rapport avec le problème PERT n'est pas assez documentée. Ce rapport essaie d'apporter quelques pistes qu'il faut explorer davantage. De plus, le réseau PERT présenter ici n'utilise que la notion de ressource de travail (Équipes). En effet on pourrait avoir des ressources matérielles liées à chaque tâche. Chaque jour ajouté pour la réalisation d'une tâche entrainerait une augmentation du coût total du projet [4]. Le préordre leximin permet de traiter les problèmes d'optimisation tout en respectant un sens d'équité. Et la programmation par contraintes constitue un framework fort pour ce type d'optimisation multiobjectif. La méthode implémentée dans ce rapport est naïve et ne fonctionne pas sur des projets de grande ampleur.

Néanmoins, cette idée d'optimisation multicritère serait pertinente pour certains softwares de gestion de projet. Par exemple, on peut mentionner MS Projet qui ne contient pas ce type d'option.

Finalement, même si MMF et leximin sont des notions essentielles lorsqu'on veut traiter de l'allocation juste des ressources, il serait intéressant de les remplacer par une fonction paramétrée qui équilibrerait entre égalitarisme et utilitarisme puis comparer ces résultats.

Références

[1] Bouveret, Sylvain and Michel Lemaître. "Comparison of two constraint programming algorithms for computing leximin-optimal allocations." (2006).

[2] Contributeurs de Wikipédia, "PERT," Wikipédia, l'encyclopédie libre

[3] Nace, Dritan, and James B. Orlin. "Lexicographically Minimum and Maximum Load Linear Programming Problems." Operations Research 55, no. 1 (2007) : 182-87

[4] Agyei, Wallace. "Project Planning And Scheduling Using PERT And CPM Techniques With Linear Programming : Case Study." International Journal of Scientific & Technology Research 4 (2015) : 222-227.

[5] Nace, Dritan & Doan, Linh & Klopfenstein, Olivier & Bashllari, Alfred. (2008). Max-Min Fairness in multi-commodity flows. Computers & Operations Research. 35. 557-573. 10.1016/j.cor.2006.03.020.