Gestion de production et ressources humaines

Méthodes de planification dans les systèmes productifs

18 mars 2005

Table des matières

Chapitre 4

Gestion de Projet

Christian Artigues et Sophie Demassey

Un projet est constitué d'un ensemble de tâches à réaliser au moyen d'un nombre limité de ressources et suivant un ou plusieurs objectifs donnés. L'ordonnancement du projet consiste à déterminer les dates d'exécution des tâches en tenant compte de la disponiblité des ressources et de façon à satisfaire au mieux les objectifs fixés. La souplesse de cette définition engendre une grande variété de modèles parmi les plus généraux de l'ordonnancement. Les problèmes d'ordonnancement de la production ou de génération d'horaires, par exemple, peuvent être vus comme des cas particuliers de l'ordonnancement de projet. D'autres applications se rencontrent aussi dans le domaine de l'électronique et l'ordonnancement de processus en informatique sous contraintes d'utilisation de ressources de type processeurs, mémoire, périphériques.

4.1 Situation de l'ordonnancement en gestion de projet

Comme en production ou dans les services, la gestion de projet comprend les différentes étapes pour parvenir à la réalisation d'un projet, et même au-delà, avec les processus de vérification et de contrôle de la réalisation.

La définition du projet constitue la première étape. Il s'agit d'identifier précisément tous les composants du projet : les *ressources* (machines, main d'oeuvre, équipement,

matériaux, budget, etc.) et les *tâches*, ou *activités*, à effectuer.

Les spécifications techniques basées sur l'analyse des contraintes technologiques ou financières du projet caractérisent les tâches et les ressources (nombre, capacité et qualification). Elles permettent aussi d'identifier un ordre obligatoire dans la réalisation de certaines tâches et de lier les tâches aux ressources, en déterminant la quantité de chacune des ressources nécessaire à l'exécution des tâches. Elles permettent enfin de définir le ou les critères de performance selon lesquels les différents ordonnancements possibles du projet seront évalués.

Au niveau opérationnel, une fois les caractéristiques du projet connues ou du moins prévues, les décisions prises consistent à déterminer un ordonnancement réalisable des tâches qui satisfait au mieux les critères de performance. Des travaux pour la résolution de ces problèmes ont été menés dans différents domaines de l'optimisation combinatoire avec des approches issues de la recherche opérationnelle ou de la programmation par contraintes.

La complexité algorithmique de ce type de problèmes et la taille des projets traités en pratique sont telles qu'on utilise généralement des *méthodes approchées*, ou *heuristiques* permettant de calculer en un temps raisonnable de «bons» ordonnancements, contrairement aux *méthodes exactes*, qui prouvent, si elles arrivent à leur terme, l'optimalité du meilleur ordonnancement calculé. D'autres approches de résolution prennent en compte les incertitudes dans la définition du projet : l'*approche stochastique* est employée quand les données du problème prennent des valeurs probabilistes ; l'*approche robuste* consiste à proposer des ordonnancements restant admissibles si les données varient.

La dernière phase de la gestion du projet consiste alors à la réalisation même du projet, suivant l'ordonnancement choisi et en tenant compte éventuellement des aléas lors de la réalisation.

4.2 Ordonnancement de projet sous contraintes de ressources

4.2.1 Définition et notations

Le *problème d'ordonnancement de projet sous contraintes de ressources*, aussi appelé *problème d'ordonnancement de projet à moyens limités*, ou encore *RCPSP* pour «*resource-*

constrained project scheduling problem», désigne ici le problème de référence en gestion de projet.

Les définitions et notations en ordonnancement du chapitre 2 s'appliquent encore à ce problème, à la différence qu'une ressource ici (contrairement à une machine dans un problème de production) est cumulative. C'est à dire qu'elle permet d'exécuter plusieurs tâches simultanément, à condition que la quantité de ressource requise par les tâches (la somme de leurs consommations) n'excède pas la quantité limite disponible de la ressource (sa capacité). Il est à noter aussi que le terme de projet correspond, en production, au terme de travail tandis qu'une tâche ici correspond à une opération. Une instance du RCPSP se caractérise formellement comme suit :

- i indice des tâches, $i \in \{0, 1, \ldots, n, n+1\}$. 0 et $n+1$ désignent des tâches fictives modélisant respectivement le début et la fin du projet ;
- p_i durée de la tâche i. La préemption n'est pas autorisée : chaque tâche i s'exécute durant exactement p_i unités de temps consécutives ;
- E relation d'ordre partiel sur l'ensemble des tâches ;
- k indice des ressources $k \in \{1, 2, \ldots, m\}$;
- R_k capacité de la ressource k. Les ressources sont renouvelables : la quantité R_k est constante sur toute la durée de l'ordonnancement ;
- r_{ik} consommation de la tâche i sur la ressource k à chaque instant de son exécution ;
- T horizon (ou durée maximale) d'ordonnancement.

En notant S_i la date de début d'exécution de la tâche i, avec la convention $S_0 = 0$, le RCPSP consiste à déterminer un ordonnancement de durée minimale, autrement dit, un vecteur $S = (S_0, S_1, \ldots, S_n + 1)$ qui satisfasse :

- les contraintes de précédence entre les tâches : $(i, j) \in E$ indique que la tâche j ne peut commencer son exécution avant l'achèvement de la tâche i :

$$S_j \geq S_i + p_i \quad \forall (i, j) \in E \tag{4.1}$$

- les contraintes de ressources, indiquant qu'à chaque instant t, la quantité d'une ressource k utilisée par l'ensemble $\mathcal{A}_t$ des tâches en cours d'exécution, n'excède pas la capacité de k :

$$\sum_{i \in \mathcal{A}_t} r_{ik} \leq R_k \quad \forall t = 0, \ldots, T, \forall k = 1, \ldots, m \tag{4.2}$$

  avec $\mathcal{A}_t = \{i = 1, \ldots, n \mid S_i \leq t < S_i + p_i\}$.
- la durée du projet S_{n+1} est minimale et égale à T dans le pire des cas.

Habituellement, un projet est représenté par un graphe orienté G, le *graphe potentiels-tâches*, où les noeuds identifient les tâches du projet et où, à tout couple (i, j) de E, est associé un arc $i \to j$ valué par la durée p_i.

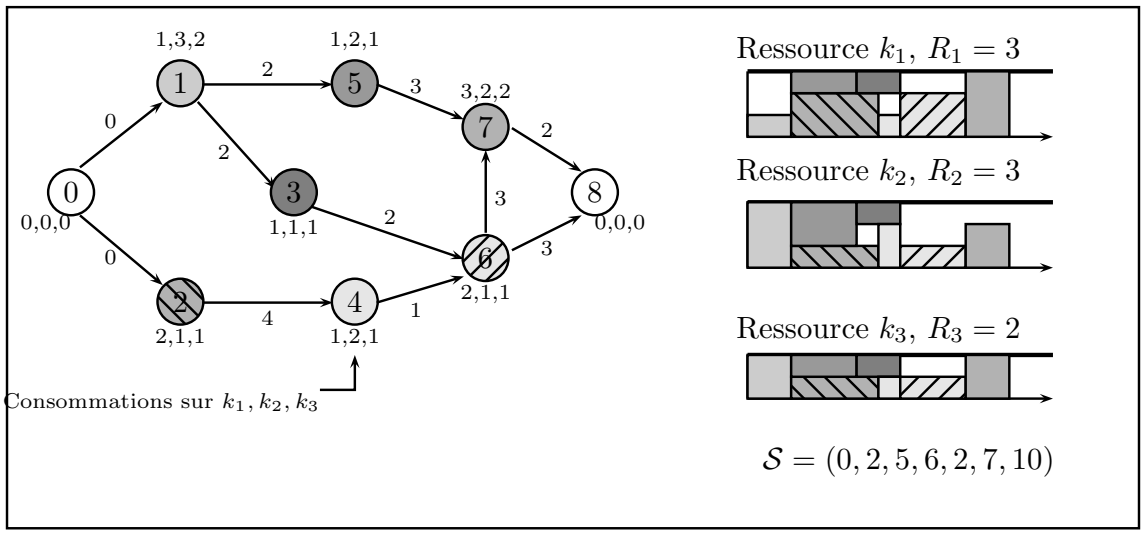

FIG. 4.1 – *Exemple d'instance du RCPSP.*

La figure 4.1 représente une instance simple du RCPSP à 7 tâches et 3 ressources (exemple tiré de [28]). Les noeuds du graphe potentiels-tâches sont valués ici par les consommations des tâches sur les ressources. Une solution du problème, de durée optimale égale à 12, est représentée à droite par le *diagramme de Gantt* d'occupation des ressources en fonction du temps.

Le RCPSP appartient à la classe des problèmes combinatoires les plus difficiles, mais les contraintes du problème interviennent différemment dans cette difficulté. En fait, le problème d'ordonnancement sous contraintes d'une unique ressource et sans contrainte de précédence est NP-difficile au sens fort [19]. Au contraire, sans contraintes de limitation des ressources, le problème est un cas particulier du *problème central de l'ordonnancement* et se résoud en temps polynomial.

4.2.2 Contraintes de précédence

Le problème central de l'ordonnancement consiste à ordonnancer un ensemble de tâches soumises à des contraintes temporelles de type inégalités de potentiels $S_j - S_i \geq a_{ij}$, avec $a_{ij} \in \mathbb{R}$, en minimisant la date de fin de l'ordonnancement [32]. On considère dans

cette section, un cas particulier[1], celui du RCPSP sans contrainte de ressources, où les inégalités de potentiels ne sont définies que pour les couples $(i, j) \in E$ et en posant $a_{ij} = p_i$.

Une condition nécessaire et suffisante pour qu'il existe une solution à ce problème est que le graphe potentiels-tâches G ne possède pas de circuit. Une solution S de durée minimale est déterminée en posant, pour chaque tâche i, S_i égal à la longueur des plus longs chemins de 0 à i dans le graphe G [11]. Cette solution est appelée l'*ordonnancement au plus tôt* et notée $ES = (ES_0, \ldots, ES_{n+1})$ pour *Earliest Starting times*. ES_i représente alors la *date de début au plus tôt* de la tâche i.

Si la date T de fin au plus tard de l'ordonnancement est prise en compte et si $T < ES_{n+1}$, alors le problème n'a pas de solution. Sinon, de manière symétrique, on obtient la *date de fin au plus tard* LS_i d'une tâche i en soustrayant à T la longueur d'un plus long chemin de i à $n+1$. $LS = (LS_0, \ldots, LS_{n+1})$ est appelé l'*ordonnancement au plus tard* (*Latest Starting times*).

Plus généralement, le calcul d'un plus long chemin dans le graphe G entre deux tâches i et j permet de déterminer la *distance minimale* d_{ij} telle que les dates de début des tâches vérifient $S_j - S_i \geq d_{ij}$. En effet, l'exécution de la tâche j ne peut démarrer tant que toutes les tâches situées sur un chemin reliant i à j dans le graphe G ne sont pas terminées.

La connaissance des dates de début au plus tôt et au plus tard donne des renseignements précieux sur les *marges* dont on dispose pour réaliser le projet, en l'absence de contraintes de ressources. Les tâches dont la marge totale $LS_i - ES_i$ est nulle sont situées sur les plus longs chemins de 0 à $n+1$. Ces chemins sont appelés *chemins critiques*, et les tâches qui le constituent sont appelées *tâches critiques*.

Pour déterminer les dates, les marges et le chemin critique, plusieurs méthodes ont été proposées. La méthode MPM est celle évoquée ci-dessus, basée sur le graphe potentiels-tâches. D'autres méthodes, comme la méthode PERT (Program Evaluation and Review Technique) et CPM (Critical Path Method) sont basées sur un autre graphe, appelé potentiels-étapes, où les tâches sont représentées par les arcs. Toutes ces méthodes reposent sur des algorithmes de calcul de plus longs chemins. L'algorithme de Bellman permet par exemple de calculer tous les plus long chemin d'un sommet à tous les autres

[1]Une généralisation du RCPSP (RCPSP avec fenêtres de temps ou RCPSP/max) correspond exactement au problème central de l'ordonnancement augmenté des contraintes de ressources

en $\mathcal{O}(|E|)$[2]. Il permet donc de déterminer les dates de début au plus tôt et au plus tard. Le calcul des distances minimales d_{ij} peut être effectué par l'algorithme de Floyd-Warshall qui détermine les plus longs chemins de tous les sommets vers tous les autres, en $\mathcal{O}(n^3)$.

Une solution qui respecte les contraintes de ressources d'une instance du RCPSP est nécessairement une solution réalisable du problème central de l'ordonnancement sous-jacent, obtenu en supprimant les contraintes de ressources. La valeur optimale de ce sous-problème est donc une évaluation par défaut de la valeur optimale du RCPSP initial. En fait, pour chaque tâche i, ES_i (resp. LS_i) est une évaluation par défaut (resp. par excès) de la date de début de i dans tout ordonnancement réalisable du RCPSP. De même, d_{ij} (resp. $-d_{ji}$) est une évaluation par défaut de la distance $S_j - S_i$.

4.2.3 Programmation par contraintes pour le RCPSP

Les principes généraux de la programmation par contraintes et des applications à l'ordonnancement sont présentés dans le chapitre 8. Dans cette section, nous présentons tout d'abord un modèle de satisfaction de contraintes du RCPSP et des techniques de maintien de la cohérence pour l'ordonnancement qui s'appliquent à ce modèle. Nous évoquons ensuite des algorithmes de filtrage plus spécifiques au cas cumulatif du RCPSP.

4.2.3.1 Modèle de satisfaction de contraintes

Les équations (4.1) et (4.2) sont valides dans un modèle de satisfaction de contraintes pour le RCPSP. Alternativement, dans la formulation suivante, les contraintes sur chacune des ressources sont modélisées au moyen de la contrainte globale `cumulative` (4.4), décrite dans le chapitre 8.

$$S_j - S_i \geq p_i \qquad \forall\ (i,j) \in E \tag{4.3}$$

$$\texttt{cumulative}(< S_1, \ldots, S_n >, < p_1, \ldots, p_n >, < r_{1k}, \ldots, r_{nk} >, R_k)\ \forall\ k = 1, \ldots, m \tag{4.4}$$

$$S_i \in \{0, \ldots, T\} \qquad \forall\ i = 0, \ldots, n+1 \tag{4.5}$$

Pour assurer une complexité algorithmique raisonnable, généralement seules les bornes des domaines des variables S_i sont maintenues *cohérentes*. En particulier, maintenir la

[2]Ce résultat étant valable uniquement si G est sans circuit

cohérence d'arc aux bornes sur les contraintes de précédence (4.3) revient à calculer les dates de début au plus tôt et au plus tard dans le graphe de précédence : $S_i \in \{ES_i, \ldots, LS_i\}$. Comme indiqué dans la section 4.2.2, elle peut ainsi être assurée en $\mathcal{O}(|E|)$ par l'algorithme de Bellman.

La modélisation des demandes sur les ressources par la contrainte globale `cumulative` permet d'exploiter les algorithmes de filtrage associés à cette contrainte, en particulier les règles issues du raisonnement énergétique (ces règles sont présentées au chapitre 5). Par une étude expérimentale, Baptiste *et al.* [6] ont mis à jour des dominances entre différentes règles d'ajustement applicables au RCPSP. Ils montrent notamment que le comportement du raisonnement énergétique dépend clairement du type d'instances testées. En effet, son coût en temps de calcul se ressent fortement sur les instances *hautement disjonctives* où un grand nombre de tâches sont incompatibles deux à deux. Au contraire, il prouve son efficacité sur les instances *hautement cumulatives*.

4.2.3.2 Contraintes redondantes et problèmes à une machine

Les règles de *Edge-Finding* appartiennent à une seconde classe d'algorithmes de filtrage associés à la contrainte `cumulative`. En réalité, ce type d'algorithmes s'applique plus généralement, à tout problème à une machine et donc à tout sous-problème à une machine extrait d'un problème d'ordonnancement cumulatif tel que le RCPSP.

Un ensemble C de tâches ne pouvant être exécutées en parallèle deux à deux est appelé *ensemble disjonctif* (par exemple, un ensemble où, pour chaque paire de tâches (i, j), il existe une ressource k telle que $r_{ik} + r_{jk} > R_k$). Associer un tel ensemble C à une machine (ressource disjonctive) utilisée par toutes les tâches de C revient à définir une contrainte redondante par rapport au modèle de base. Cette redondance n'est toutefois pas inutile.

L'algorithme du Edge-Finding permet de déterminer des séquencements relatifs parmi les tâches d'un tel ensemble C et effectue les ajustements associés. Supposons en effet que l'exécution de C se termine (au plus tôt à la date $ES_C = \min_{j \in C} ES_j + \sum_{j \in C} p_j$) après la date de fin au plus tard de toutes les tâches de C exceptée une tâche i ($ES_C > \max_{j \in C \setminus \{i\}}(LS_j + p_j)$), alors i est nécessairement la tâche de C exécutée en dernier. Sa date de début au plus tôt peut ainsi être mise à jour, sachant alors que :

$$ES_i \geq \max_{C' \subseteq C} ES_{C'}.$$

De manière symétrique, le Edge-Finding permet de déterminer si une tâche doit être exécutée avant toutes les tâches de C. Pour un ensemble C donné, tous les ajustements de ce type peuvent être effectués en $\mathcal{O}(n^2)$. Nous renvoyons de nouveau le lecteur à [6] pour la description de l'algorithme qui effectue ces ajustements et pour la présentation d'autres algorithmes de filtrage pour les problèmes à une machine (telles que les règles *Not First/Not Last*).

Ces règles ont prouvé leur efficacité pour la résolution de problèmes d'ordonnancement disjonctifs, notamment le job-shop [14]. Cependant une difficulté supplémentaire ici est d'identifier les ensembles disjonctifs induits par le RCPSP. Inversement, certaines instances du RCPSP peuvent ne comporter que peu de tâches incompatibles deux à deux, ce qui empêche alors d'utiliser ce type de règles.

4.2.3.3 Autres algorithmes de filtrage

D'autres algorithmes de filtrage ont été adaptés à la résolution du RCPSP, notamment : les règles sur les triplets symétriques [10] ou encore, la technique de cohérence globale du *shaving* [16]. Ces techniques permettent non seulement d'ajuster les bornes des domaines des variables, mais aussi de détecter de nouvelles *disjonctions* (ou couples de tâches incompatibles) qui, dans un processus de propagation de contraintes, peuvent à leur tour être prises en compte dans la génération d'ensembles disjonctifs, puis dans l'application des règles sur les problèmes à une machine.

4.2.4 Programmation linéaire pour le RCPSP

Les travaux les plus anciens menés sur la résolution exacte du RCPSP font appel à la programmation linéaire en nombres entiers. La principale difficulté ici consiste à modéliser les contraintes de ressources (4.2) au moyen d'inégalités linéaires. Les différentes formulations linéaires du RCPSP souffrent ainsi soit de leur taille (large nombre de contraintes ou de variables) soit de la faiblesse de leurs relaxations. Pour traiter des instances de taille raisonnable (au moins 60 tâches), l'application de techniques de programmation linéaire bien spécifiques est alors nécessaire (section 4.2.6).

On distingue principalement deux sortes de formulations : celles en *en temps continu* sont basées sur des variables binaires x_{ij} définies par $x_{ij} = 1$ si et seulement si la tâche i précède la tâche j, et celles en *en temps discrétisé* reposent sur des variables binaires

indicées à la fois par les tâches et les instants. Nous donnons trois formulations standard pour le problème et nous terminons ce paragraphe par la présentation de quelques techniques de prétraitement, qui permettent de réduire la taille des programmes linéaires, et de génération d'inégalités valides, qui permettent de renforcer les formulations.

4.2.4.1 Formulation en temps continu basée sur les ensembles critiques

Alvarez-Valdès et Tamarit [1] se basent sur la formulation disjonctive de Balas [4] et la notion d'*ensemble critique minimal* de tâches $C \in \mathcal{C}m$. Un ensemble C de tâches est critique si la somme des consommations des tâches de C excède la quantité disponible pour au moins une ressource, et critique minimal si aucun sous-ensemble strict de C n'est critique.

La formulation suivante en temps continu est basée sur la propriété que dans toute solution réalisable, au moins deux tâches dans chaque ensemble admissible, ne peuvent être exécutées en parallèle (contraintes 4.9).

$$\min S_{n+1} \tag{4.6}$$

sujet à :

$$x_{ij} = 1 \qquad \forall\ (i,j) \in E \tag{4.7}$$

$$S_j - S_i \geq -M + (p_i + M)x_{ij} \qquad \forall\ i,j = 0,\ldots,n+1 \tag{4.8}$$

$$\sum_{(i,j)\in C^2} x_{ij} \geq 1 \qquad \forall\ C \in \mathcal{C}m \tag{4.9}$$

$$x_{ij} \in \{0,1\},\ x_{ii} = 0, S_i \geq 0 \qquad \forall\ i,j = 0,\ldots,n+1 \tag{4.10}$$

(4.7) modélise les contraintes de précédence initiales du problème. Les contraintes (4.8) lient les deux types de variables du modèle : pour une valeur de M suffisament grande (par exemple $M = T$), les contraintes imposent que, pour toute précédence $x_{ij} = 1$, la tâche j doit débuter son exécution après la complétion de i. Autrement, si $x_{ij} = 0$, la distance $S_j - S_i$ n'est pas contrainte ($S_j - S_i \geq -M$). Cette formulation contient un nombre exponentiel de contraintes (4.9) et l'ensemble $\mathcal{C}m$ n'est donc pas totalement énuméré en pratique. De plus, la présence des contraintes «grand-M» (4.8) affaiblit la relaxation continue de ce programme.

4.2.4.2 Formulation classique en temps discretisé

Les formulations en temps discretisé contiennent un nombre de variables dépendant de l'horizon T. Si les relaxations obtenues sont de meilleure qualité que pour les modèles en temps continu, la discrétisation du temps engendre des programmes de grande taille dans le cas de certaines instances.

La première modélisation du RCPSP en un programme linéaire a été donnée par Pritsker *et al.* [30]. Une variante est reportée ici avec les variables binaires : $y_{it} = 1$ si la tâche i débute à l'instant t et 0 sinon. L'ensemble des tâches en cours à un instant t s'écrit alors $\mathcal{A}_t = \{i = 1, \ldots, n \mid \sum_{s=t}^{t+p_i-1} y_{is} = 1\}$, ce qui permet de traduire les contraintes de ressources (4.2) par les inégalités (4.14). Les contraintes de précédence (4.13) et de non-préemption (4.12) et l'objectif (4.11) s'expriment clairement avec la correspondance $S_i = \sum_{t=0}^{T} y_{it}$.

$$\min \sum_{t=0}^{T} t y_{(n+1)t} \tag{4.11}$$

sujet à :

$$\sum_{t=0}^{T} y_{it} = 1 \qquad \forall\ i = 0, \ldots, n+1 \tag{4.12}$$

$$\sum_{t=0}^{T} t(y_{jt} - y_{it}) \geq p_i \qquad \forall\ (i,j) \in E \tag{4.13}$$

$$\sum_{i=1}^{n} r_{ik} \sum_{\tau=t-p_i+1}^{t} y_{i\tau} \leq R_k \qquad \forall\ k = 1, \ldots, m,\ \forall\ t = 0, \ldots, T \tag{4.14}$$

$$y_{it} \in \{0, 1\} \qquad \forall\ i = 0, \ldots, n+1,\ \forall\ t = 0, \ldots, T \tag{4.15}$$

4.2.4.3 Formulation en temps discrétisé basée sur les ensembles admissibles

Tout ordonnancement réalisable est clairement caractérisé par la séquence $(\mathcal{A}_0, \ldots, \mathcal{A}_T)$ des ensembles de tâches exécutées à chaque instant. Par exemple, l'ordonnancement représenté sur la figure 4.2 peut s'écrire alternativement par les dates de début des tâches $S = (0, 0, 2, 2, 4, 5, 6, 8)$ ou par la suite des ensembles de tâche exécutés à chaque instant $(\{1,2\}, \{1,2\}, \{2,3,4\}, \{3,4\}, \{4,5\}, \{5,6\}, \{5,7\}, \{7\}, \{8\})$. Mingozzi *et al.* [26] ont construit leur modèle en temps discretisé sur cette caractérisation et sur la notion d'*ensemble admissible*.

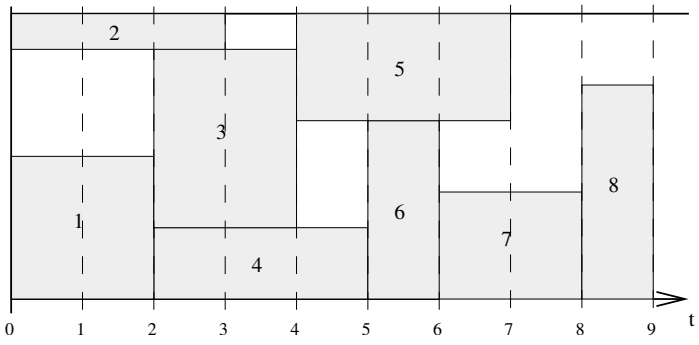

FIG. 4.2 – *Ordonnancements et ensembles admissibles.*

Les ensembles $\mathcal{A}_t$ sont dits admissibles car les tâches qui les composent sont autorisées à s'exécuter simultanément au vu des contraintes de précédence et de ressources.

Un ensemble admissible est noté ci-dessous F_l avec l appartenant dans l'ensemble des indices $\mathcal{F}$. Pour toute tâche i, $\mathcal{F}_i$ désigne le sous-ensemble des indices des ensembles admissibles contenant la tâche i.

Ce modèle est identique au précédent, exceptées les contraintes de ressources (4.14) modélisées ici par les contraintes (4.16) à (4.19). Ces contraintes portent sur des variables de décision x_{lt} définies pour tout ensemble admissible F_l et pour tout temps t : $x_{lt} = 1$ si toutes les tâches de F_l sont en cours d'exécution au temps t. Il est à noter que ce programme linéaire contient ainsi un nombre exponentiel de variables.

Les contraintes (4.17) n'autorisent au plus qu'un ensemble admissible à être en cours d'exécution à un temps t. Les contraintes (4.16) garantissent que toute tâche i s'exécute pendant p_i unités de temps et les contraintes (4.18), que i commence au temps t si i appartient à l'ensemble admissible en cours au temps t mais pas à celui en cours à $t-1$.

$$\sum_{l \in \mathcal{F}_i} \sum_{t=0}^{T} x_{lt} = p_i \qquad \forall\, i = 1, \ldots, n \tag{4.16}$$

$$\sum_{l \in \mathcal{F}} x_{lt} \leq 1 \qquad \forall\, t = 0, \ldots, T \tag{4.17}$$

$$y_{it} \geq \sum_{l \in \mathcal{F}_i} x_{lt} - \sum_{l \in \mathcal{F}_i} x_{lt-1} \qquad \forall\, t = 0, \ldots, T,\ \forall\, i = 1, \ldots, n \tag{4.18}$$

$$x_{lt} \in \{0, 1\},\ x_{l(-1)} = 0 \qquad \forall\, l \in \mathcal{F}, \forall\, t = 0, \ldots, T \tag{4.19}$$

4.2.4.4 Prétraitements des programmes linéaires et inégalités valides

En programmation linéaire, les techniques de prétraitement ont pour but d'éliminer des variables et de renforcer les coefficients des contraintes. Les techniques les plus récentes utilisent les déductions du filtrage sur la formulation de programmation par contraintes.

Dans un modèle en temps discrétisé, par exemple, il est possible d'annuler toute variable y_{it} telle que $t < ES_i$ ou $t > LS_i$. Un filtrage efficace permet d'affiner les dates de début au plus tôt et au plus tard et ainsi d'éliminer de nombreuses variables du modèle linéaire.

Les inégalités (4.8) du modèle en temps continu sont réputées donner de mauvaises relaxations. Elles peuvent être néanmoins renforcées en remplacant les coefficients M, avec les valeurs calculées (et éventuellement resserrées par propagation de contraintes) des distances minimales d_{ij} (§ 4.2.2). Les contraintes (4.8) peuvent ainsi être réécrites :

$$S_j - S_i \geq d_{ij} + (p_i - d_{ij})x_{ij} \qquad \forall\ i, j = 0, \ldots, n+1 \tag{4.8'}$$

Comme pour le prétraitement, les déductions de la propagation de contraintes peuvent aussi être utiles à la génération d'inégalités valides pour resserer la relaxation d'un modèle linéaire. Il est aussi possible de traduire des contraintes redondantes en programmation par contraintes sous la forme d'inégalités linéaires. Si par exemple, l'algorithme du Edge-Finding (§ 4.2.3) indique qu'une tâche l doit précéder toutes les tâches d'un ensemble disjonctif C, on peut alors générer une inégalité valide sur la distance minimale $S_j - S_l$ entre une tâche j de C et l. En introduisant les variables x_{ij} pour les autres tâches i de l'ensemble C on sait que $S_j - S_l \geq \sum_{i \in C \setminus \{j\}} p_i x_{ij}$ puisque l est avant toutes les tâches de C. De plus on peut éventuellement augmenter encore cette distance en ajoutant la plus petite des distances minimales entre la fin de l et les autres tâches de C. On obtient ainsi :

$$S_j \geq S_l + \sum_{i \in C \setminus \{j\}} p_i x_{ij} + \min_{i \in C \setminus \{l\}} (d_{li} - p_l) \quad \forall j, l \in C \text{ t.q. } l \to C \setminus \{l\}. \tag{4.20}$$

Des présentations plus complète des techniques de prétraitement et de génération de coupes basés sur les algorithmes de filtrage peuvent être trouvées dans [16] et [5].

4.2.5 Heuristiques et métaheuristiques

4.2.5.1 Méthodes de construction progressive

Algorithmes sériel et parallèle Le moyen le plus simple de construire une solution au problème consiste en l'application d'une méthode constructive. La plupart des méthodes constructives partent du même principe et comportent n étapes, une tâche étant ordonnancée (sa date de début est définitivement fixée) à chaque étape. L'ordre dans lequel les tâches sont sélectionnées est en général déterminé par une règle de priorité. La date de début de la tâche sélectionnée est déterminée par l'algorithme d'ordonnancement. Nous présentons les deux principaux algorithmes d'ordonnancement par construction progressive : l'*algorithme sériel* et l'*algorithme parallèle*. Dans les deux cas, ces deux algorithmes prennent en entrée une liste de tâches classées dans un ordre compatible avec les contraintes de précédence (aucune tâche ne peut se trouver classée après un de ses successeurs). Ces algorithmes sont à rapprocher des approches distribuées statiques décrites (ou approches par simulation) dans le chapitre 2. La différence principale est qu'on ne se situe plus ici dans le cadre d'une approche distribuée car une tâche peut utiliser plusieurs ressources simultanément. Le point commun est qu'il s'agit bien de simuler l'évolution de l'exécution du projet dans l'ordre topologique ou chronologique pour les algorithmes sériel ou parallèle, respectivement.

L'algorithme sériel sélectionne les tâches dans l'ordre de la liste et ordonnance la tâche le plus tôt possible compte tenu des contraintes de précédence et de ressources, en tenant compte des tâches ordonnancées aux étapes précédentes.

L'algorithme parallèle ordonnance les tâches en incrémentant un instant de décision t initialisé à 0. A chaque étape, l'algorithme parcourt la liste des tâches non encore ordonnancées et place successivement au temps t les tâches possibles, sans violer de contraintes de précédence ou de ressources. Quand plus aucune tâche ne peut démarrer à t, t est augmenté d'une unité.

Nous illustrons le fonctionnement de ces algorithmes sur le petit exemple décrit à gauche de la figure 4.3 qui comporte 7 tâches réelles et une ressource de capacité 5. En sélectionnant les tâches dans l'ordre de priorité $< 0, 1, 5, 4, 6, 3, 7, 2, 8 >$, on obtient l'ordonnancement de gauche avec l'algorithme sériel et l'ordonnancement de droite avec l'algorithme parallèle. Les deux algorithmes génèrent des ordonnancement actifs, c'est-à-dire qu'aucune tâche ne peut être démarrée plus tôt sans déplacer les autres tâches. L'algorithme parallèle génère en plus des ordonnancements sans délai, c'est-à-dire qu'à

aucun moment la ressource n'est laissée inactive alors qu'elle pourrait démarrer l'exécution d'une tâche (dont tous les prédécesseurs sont achevés). Ainsi, la tâche 3 pourrait être démarrée à la date 0 dans l'ordonnancement de gauche. Cet ordonnancement n'est donc pas sans délai.

La théorie de l'ordonnancement nous indique qu'il existe obligatoirement un ordonnancement actif optimal, que l'algorithme sériel est ainsi susceptible de trouver si on lui donne la bonne liste en entrée. Notons toutefois qu'il peut exister $n!$ listes différentes en l'absence de contraintes de précédence, ce qui, pour un nombre de tâches réalistes, exclut leur énumération totale. Par contre, rien n'indique qu'il existe un ordonnancement sans délai optimal. L'algorithme parallèle peut par conséquent se trouver dans l'impossibilité de trouver la meilleure solution, indépendamment de la liste de tâches fournie en entrée.

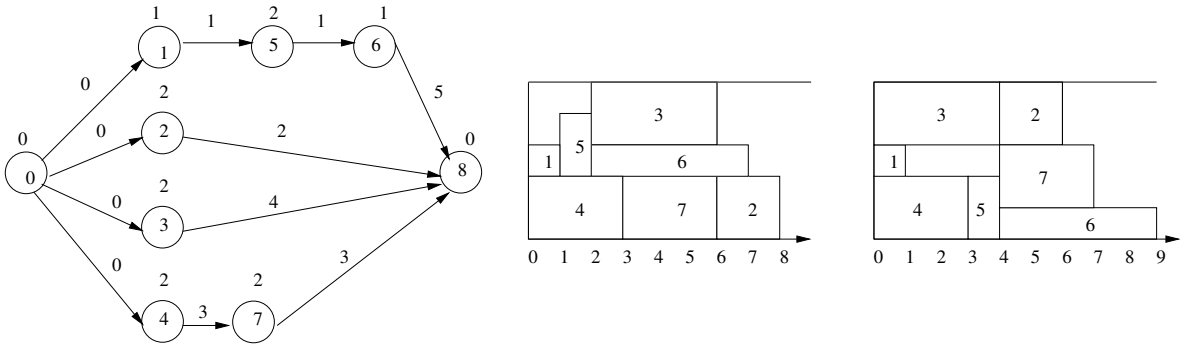

FIG. 4.3 – *Application d'algorithmes constructifs sur un RCPSP à une ressource.*

Heuristiques basées sur les règles de priorité Une heuristique simple, dite à passe unique, consiste à déterminer l'ordre de sélection des tâches par une unique règle de priorité. On peut par exemple sélectionner les tâches dans l'ordre croissant de leur marge totale (§ 4.2.2), afin d'ordonnancer les plus urgentes d'abord. La table ci-dessous donne une liste de quelques règles de priorité citées dans [24].

Les heuristiques dites à passes multiples consistent à lancer successivement l'algorithme sériel ou parallèle avec différentes listes de sélection, en conservant à tout moment la meilleure solution obtenue. On peut par exemple utiliser différentes règles de priorité ou encore, modifier aléatoirement la liste obtenue par une règle de priorité donnée. Avec cette dernière méthode, Kolisch et Hartman montrent par expérimentation sur des instances de test classiques du RCPSP [31] que les meilleurs résultats sont obtenus

TAB. 4.1 – Quelques règles de priorité pour le RCPSP.

Acronyme	tâche sélectionnée
GRPW	plus grande durée augmentée de la somme des durées des successeurs directs
LFT	plus petite date de fin au plus tard
LST	plus petite date de début au plus tard
MSLK	plus petite marge totale
MTS	plus grand nombre de successeurs directs et indirects
RSM	plus petite augmentation de la durée du projet, les tâches non sélectionnées étant décalées par l'ordonnancement de la tâche considérée
SPT	plus petite durée
WCS	plus petite marge restante si la tâche n'est pas sélectionnée

par les règles LFT et WCS, avec l'algorithme sériel pour les instances de petite taille (30 tâches), et avec l'algorithme parallèle pour les plus grandes instances (60 et 120 tâches).

Une autre méthode à passes multiples, appelée ordonnancement avant/arrière, consiste à exécuter alternativement l'algorithme constructif (sériel ou parallèle) dans l'ordre normal (*en avant*) puis, dans l'ordre inverse (*en arrière*) en inversant la liste des tâches et les arcs du graphe de précédence et en ordonnançant les tâches à partir de la fin du projet.

4.2.5.2 Metaheuristiques

De bien meilleurs résultats peuvent être obtenus par metaheuristique (section 7.3).

Un premier élément important dans la conception d'une metaheuristique est la représentation des solutions. La plupart des méthodes utilisent la propriété d'existence d'une liste donnant une solution optimale avec l'algorithme sériel et optent donc pour une représentation des solutions par listes de priorité. On trouve également une représentation proche, par *clé aléatoire*, qui consiste à affecter une valeur à chaque tâche, lui donnant ainsi une priorité.

À partir d'une liste de priorité, il est possible de définir plusieurs opérateurs de voisinage (ou opérateurs de mutation dans une méthode évolutive) comme par exemple : intervertir deux tâches ou bien décaler une tâche dans la liste (à condition que la liste obtenue respecte toujours les contraintes de précédence). Pour les méthodes évolutives, des opérateurs de croisement de deux listes peuvent également être définis : l'opérateur évoqué à la section 7.3 consiste à recopier pour moitié la liste du premier parent, puis à parcourir la liste du deuxième parent en ajoutant dans cet ordre les tâches manquantes.

Sur la base de cette représentation et de ces opérateurs, de nombreuses metaheuristiques ont été développées. Récemment, Kolisch et Hartmann [23] ont publié l'inventaire de ces méthodes, comprenant plusieurs méthodes tabou, des algorithmes génétiques, le recuit simulé ainsi que plusieurs méthodes hybrides. Les auteurs concluent, sur la base de résultats expérimentaux, que les meilleures méthodes sont celles mélangeant divers ingrédients, comme les méthodes évolutives hybridées avec de la recherche tabou [22] ou avec des méthodes d'ordonnancement avant/arrière [33].

D'autres metaheuristiques récentes tentent d'explorer de nouvelles représentations des solutions (par les schémas d'ordonnancements disjonction-parallélisme [3] ou par les flots d'unités de ressources [2]) ou bien, de nouvelles formes d'hybridation au sein de méthodes de grand voisinage, où la recherche du meilleur voisin est un problème NP-difficile, résolu par recherche arborescente tronquée [29].

4.2.6 Méthodes exactes pour le RCPSP

Étant donnée la complexité algorithmique du RCPSP, la recherche arborescente est la base des méthodes dédiées à la résolution exacte de ce problème.

Les différentes méthodes de la littérature se distinguent donc principalement par le choix d'un schéma de branchement (qui régit la séparation de l'espace de recherche) et le choix de la méthode d'évaluation (calculs de bornes inférieures ou algorithmes de filtrage).

Les principales procédures arborescentes pour le RCPSP sont recensées dans plusieurs états de l'art récents [7, 18].

4.2.6.1 Schémas de branchement et règles de dominance

Les schémas de branchement de la littérature du RCPSP sont majoritairement de type chronologique. Ils reposent sur la construction progressive de séquencements partiels, à la manière des algorithmes parallèle et sériel (section 4.2.5). À chaque noeud de l'arbre de recherche, le séquencement partiel associé est étendu par ajout, selon les méthodes, d'une tâche ou d'un groupe de tâches, en tenant compte des contraintes de précédence et de ressources. Ces schémas permettent ainsi l'énumération de tous les séquencements possibles. Parmi les ordonnancements au plus tôt correspondant à chacun de ces séquencements, l'un au moins est un ordonnancement optimal.

En général, les arbres chronologiques sont explorés en profondeur d'abord en choisissant d'exécuter à un instant t des tâches qui «remplissent» la ressource ou encore des tâches qui doivent être exécutées au plus vite compte tenu de leurs fenêtres de temps. Des règles de dominance permettent d'éviter d'énumérer des séquences redondantes, notamment la règle du *cut-set*, qui consiste à éliminer de la recherche, les ordonnancements partiels identifiés comme étant dominés par d'autres préalablement construits et mémorisés. Utilisant cette règle, la procédure de Demeulemeester et Herroelen [17] obtient à ce jour les meilleurs résultats sur les jeux de tests de la PSPLIB [31]. Cependant, aucune procédure n'est parvenue encore à résoudre certaines de ces instances à seulement 60 tâches.

D'autres schémas de branchement ont été proposés pour le RCPSP, s'apparentant plus clairement aux schémas de la satisfaction de contraintes.

La procédure de Carlier et Latapie [12], par exemple, calcule les fenêtres de temps de début des tâches à partir d'une durée $T-1$, T étant la durée du meilleur ordonnancement connu jusque là. T est alors optimal s'il n'existe pas d'ordonnancement réalisable dans ces fenêtres de temps. Pour prouver cela, le branchement sélectionne une tâche (une variable S_i) et coupe sa fenêtre de temps (son domaine $[ES_i, LS_i]$) en deux parties.

Brucker *et al.* [8] s'appuient sur une représentation plus particulière où chaque branchement décide du séquencement relatif pour une paire de tâches. À chaque noeud de l'arbre de recherche correspond ainsi un *schéma d'ordonnancement* (C, D, P) où C est l'ensemble des paires de tâches en conjonction (si i précède j ou bien si j précède i), D les paires de tâches en disjonction (devant s'exécuter séquentiellement mais dans un ordre encore inconnu), P les tâches s'exécutant en parallèle. À une feuille de l'arbre, quand toute paire de tâches est classée dans C, D ou P, alors il est possible de calculer

en temps polynomial le meilleur ordonnancement correspondant à un tel schéma (ou de prouver qu'il n'en existe pas).

Un autre schéma de branchement considère les ensembles interdits minimaux, sachant que dans toute solution réalisable au moins une contrainte de précédence est ajoutée entre deux tâches d'un tel ensemble (section 4.2.4). La méthode consiste ainsi à sélectionner à chaque noeud un ensemble interdit minimal, et à énumérer toutes les contraintes de précédence possibles.

4.2.6.2 Bornes inférieures

Les méthodes de résolution exacte pour le RCPSP sont principalement des procédures par séparation et évaluation, où une borne inférieure est calculée à chaque noeud de l'arbre de recherche. Outre leur rôle primordial pour limiter la taille de l'espace de recherche dans une méthode exacte, les bornes inférieures sont aussi utiles pour évaluer la qualité des solutions réalisables obtenues de manière heuristique. Le calcul de bornes inférieures pour le RCPSP peut être basé sur des considérations «spécifiques» aux problèmes d'ordonnancement, faire appel à la relaxation d'un modèle linéaire en nombres entiers, emprunter aux techniques de satisfaction de contraintes sous une forme destructive, ou bien encore combiner ces divers ingrédients.

Bornes spécifiques La borne inférieure la plus simple est appelée la borne du chemin critique et consiste à résoudre le problème d'ordonnancement sans contraintes de ressources (§ 4.2.2). Différentes extensions ont permis d'améliorer cette borne en considérant des tâches non situées sur le chemin critique mais qui ne peuvent s'exécuter en parallèle avec des tâches critiques sans violer de contraintes de ressources.

Un autre type de bornes consiste à calculer sur chaque ressource le rapport entre l'énergie totale requise par les tâches et la disponibilité de la ressource. La première de ces bornes est égale à $\max_{k\in\mathcal{R}} \left\lceil \sum_{i\in\mathcal{A}} r_{ik}p_i/R_k \right\rceil$. Des combinaisons de ces bornes avec les bornes du chemin critique ont également été utilisées.

Il est possible enfin d'extraire à partir d'une instance de RCPSP des sous-problèmes d'ordonnancement dont la résolution donne une borne inférieure. C'est le cas par exemple pour les sous-problèmes à une machine présentés dans la section 4.2.3. Carlier et Latapie [12] ont généralisé cette approche en extrayant des sous-problèmes à machines parallèles avec dates de lancement et durées de latence. La borne est alors obtenue

en résolvant, en temps polynomial au moyen de l'algorithme de Jackson préemptif, la relaxation préemptive de ces sous-problèmes.

Bornes inférieures issues de la programmation linéaire La résolution directe des programmes linéaires en nombres entiers de l'article 4.2.4, par un solveur générique n'est applicable au mieux que sur de très petites instances du RCPSP. Ces formulations ont donc généralement été présentées, accompagnées de schémas de branchement spécifiques ainsi que de techniques de relaxation adéquates.

Une relaxation consiste à supprimer les contraintes les plus difficiles d'un modèle permettant ainsi d'obtenir rapidement une borne inférieure. Pour le modèle en temps discretisé de Pritsker *et al.* [30], il suffit par exemple de supprimer les contraintes (4.15) d'intégrité des variables. Christophides *et al.* [15] ont proposé plusieurs types de coupes pour resserrer cette relaxation dans le but d'améliorer la qualité de la borne obtenue. Ces mêmes auteurs ont présenté une seconde borne issue de ce modèle, basée sur la relaxation lagrangienne des contraintes de ressources (4.14). Par la suite, au moyen d'un algorithme polynomial pour la résolution des sous-problèmes lagrangiens, Möhring *et al.* [27] ont pu obtenir un bon compromis entre la valeur de la borne et le temps de calcul.

La taille des deux autres modèles linéaires de l'article 4.2.4 rend nécessaire l'emploi de relaxations plus élaborées que la simple relaxation continue. L'une d'elles (LB3) présentée avec le modèle sur les ensembles admissibles [26] se ramène par exemple à trouver un ensemble indépendant de sommets de poid maximal dans un graphe (node packing problem). Sur ce même modèle, Brucker *et al.* [8] utilisent la génération de colonnes pour parer au nombre exponentiel de variables.

Carlier et Néron [13] ont récemment produit une borne inférieure basée sur l'énumération explicite de toutes les consommations pouvant être satisfaites simultanément pour des ressources de petite capacité. Cette borne originale présente le très net avantage d'utiliser du temps de calcul principalement à la racine de l'arbre de recherche.

Il est à noter que la plupart de ces bornes sont de bonne qualité mais sont généralement trop coûteuses en temps de calcul pour être intégrées dans une procédure de recherche arborescente. La borne (LB3) peut être calculée en $\mathcal{O}(n^3)$ et semble à ce jour offrir le meilleur compromis qualité/temps de calcul.

Bornes destructives et bornes hybrides Dans une approche destructive, les techniques de filtrage sur les contraintes permettent aussi de calculer des bornes inférieures. Il s'agit en effet, de tester la cohérence du problème (ou d'une relaxation du problème) pour différentes valeurs de l'horizon T prises, par exemple, par dichotomie sur un intervalle. Une borne inférieure est donnée par la plus petite valeur de T pour laquelle on ne peut prouver que le problème de satisfaction est incohérent.

Les meilleures bornes inférieures (Brucker et Knust [10], Demassey *et al* [16], et Baptiste et Demassey [5]) connues à ce jour sur les jeux de test de la PSPLIB [31] combinent à la fois propagation de contraintes, programmation linéaire et approche destructive.

4.3 Extensions

Nous présentons les extensions les plus courantes du modèle de base présenté dans le paragraphe précédent pour prendre en compte la diversité des objectifs et des caractéristiques pratiques des tâches et des ressources. Les méthodes existantes pour le problème de RCPSP «classique» doivent bien sûr être adaptées à ces nouvelles caractéristiques. Nous n'aborderons pas ici ces méthodes mais renvoyons le lecteur à l'ouvrage de Demeulemeester et Herroelen [18] où les variantes du problème d'ordonnancement de projet et les différentes méthodes sont décrites en détails.

4.3.1 Fonctions objectif

La minimisation de la durée totale du projet n'est pas la seule fonction présente dans la réalité des projets. De nombreuses autres fonctions objectif permettent de prendre en compte la nécessité d'équilibrer l'utilisation des ressources, de gérer un budget alloué au projet ou de prendre en compte les concepts du *juste-à-temps*.

4.3.1.1 Gestion des ressources

La durée du projet représente un objectif lié à l'exécution des tâches. D'autres objectifs réalistes sont liés à la gestion des ressources. On peut par exemple souhaiter que le profil d'utilisation des ressources soit le plus équilibré possible. Il s'agit du problème de lissage des ressources (*resource levelling problem*). Dans ce cas, une date limite T est imposée au projet ($S_{n+1} \leq T$) et l'utilisation maximale à un instant donné d'une ressource

est une variable (R_k est alors remplacée par R_{kt}). La fonction objectif est remplacée par $\min \sum_{k=\mathcal{R}} \sum_{t=1}^{t=T} f_k(R_{kt})$ où f_k est une fonction dépendant des applications (par exemple $f_k(R_{kt}) = w_k R_{kt}^2$ vise à minimiser la somme pondérée du carré des utilisations des ressources).

Les ressources disponibles pour un projet ne sont souvent pas disponibles avant l'exécution du projet et doivent être achetées ou louées (pour le matériel) ou embauchées (pour le personnel). Il existe donc un coût d'utilisation des ressources dépendant de la quantité demandée. On peut chercher à minimiser ce coût en considérant que la quantité totale de chaque ressource R_k est une variable et en considérant la fonction objectif $\min \sum_{k=\mathcal{R}} c_k R_k$, minimisant le coût total d'acquisition des ressources.

4.3.1.2 Valeur actualisée nette

Il est possible de prendre en compte les flux financiers liés au projet dans la fonction objectif. Les approches les plus fréquentes considèrent qu'à chaque tâche est associé un ensemble de dépenses et de recettes qui ont lieu pendant la durée de la tâche. Soit g_{it} le flux de trésorerie associé à la tâche i à la période $t = 1, \ldots, p_i$. Le facteur d'actualisation $e^{-\alpha} = 1/(1+r)$ représente la valeur actualisée d'une unité monétaire à recevoir ou à payer à la fin de la période 1, en utilisant un taux d'actualisation r. Ainsi, à chaque tâche, on peut associer une valeur finale des flux de trésorerie qui ont lieu pendant l'exécution de i au moyen de la formule $c_i = \sum_{t=1}^{t=p_i} g_{it} e^{\alpha(d_i - t)}$. Pour maximiser la valeur actualisée nette du projet, on peut ainsi considérer la fonction objectif $\max \sum_{i \in \mathcal{A}} c_i e^{-\alpha(S_i + p_i)}$ qui actualise les flux de trésorerie des tâches en fonction de leur date de début. Notons que, dans ce modèle, le flux de trésorerie associée à une tâche est indépendant de sa date de début, ce qui est une limitation. D'autres modèles permettent de tenir compte de flux de trésorerie dépendant du temps.

4.3.1.3 Juste-à-temps

Dans beaucoup de cas pratiques, notamment dans un environnement multi-projets, les tâches ont des dates de livraison (d_i) qu'il est nécessaire de respecter au mieux. Dans le cas où l'objectif principal est de ne pas dépasser les dates de livraison des tâches, plusieurs fonctions objectifs classiques sont définies. Elles sont liées aux notion de retard algébrique $L_i = S_i + p_i - d_i$, de retard vrai $T_i = \max(0, L_i)$ et d'indicateur de retard

$U_i \in \{0,1\}$ avec $U_i = 1$ si i est en retard et $U_i = 0$ sinon. On peut ainsi minimiser le plus grand retard $L_{\max} = \max_{i\in\mathcal{A}} L_i$ ou la somme pondérée des retards vrais $\sum_{i\in\mathcal{A}} w_i T_i$.

Les environnements industriels actuels imposent souvent d'appliquer une politique de juste-à-temps où terminer un projet en avance n'est pas souhaitable, pour des raisons de stockage, de péremption ou de rentabilité. L'avance d'une tâche est donnée par $E_i = \max(0, C_i - d_i)$. On peut ainsi minimiser la somme pondérée des avances et des retards $\sum_{i\in\mathcal{A}} \alpha_i E_i + \beta_i T_i$. Cet objectif rend le problème significativement plus complexe car il supprime la propriété de dominance des ordonnancements actifs évoquée plus haut.

4.3.2 Modes d'exécution multiples

Il existe souvent plusieurs façons (modes) possibles d'exécuter une tâche en temps d'utilisation des ressources. La durée de la tâche varie selon le coût ou l'usage des ressources. La planification du projet porte alors sur des décisions de type «ordonnancement» (il faut déterminer les dates de début des tâches) mais également sur des décisions de type «affectation» (il faut choisir un mode d'exécution de la tâche). On distingue plusieurs sous-problèmes de ce type évoqués ci-après. Notons qu'un cas de modes d'exécution multiples encore plus général, non évoqué dans cette section, consiste à introduire des cheminements alternatifs dans le graphe potentiels-tâches représentant le projet, ce qui revient à permettre un choix dans les contraintes de succession. Ce principe correspond à la notion de gamme alternative de la gestion de production (chapitre 2).

4.3.2.1 Tâches à durée et coût variables

Un modèle simple pour prendre en compte les problèmes de modes multiples consiste en une extension du problème central (sans contraintes de ressources) où la durée des tâches p_i est une variable variant entre une valeur minimale $\underline{p}_i$ et une valeur maximale $\overline{p}_i$. On considère qu'à chaque activité est associé un coût qui décroit linéairement en fonction de la durée de la tâche $c_i = b_i - a_i p_i$ avec $a_i, b_i \geq 0$. Il s'agit d'un coût *direct* qui prend en compte l'utilisation des ressources. En contrepartie de ces coûts, on introduit le coût indirect du projet $C \times S_{n+1}$ qui augmente linéairement en fonction de la date de fin du projet. Il est alors nécessaire de trouver un ordonnancement qui minimise la fonction objectif donnée par $C \times S_{n+1} + \sum_{i\in\mathcal{A}}(b_i - a_i p_i)$. D'autres modèles expriment des variations plus complexes du coût en fonction de la durée.

4.3.2.2 Tâches à durée et utilisation des ressources variables

Une autre manière de prendre en compte les différentes possibilités d'exécution d'une activité consiste à prendre en compte l'utilisation des ressources (en général une seule) et à considérer que la durée de la tâche décroit de manière discrète en fonction du nombre d'unités de ressources requises. On obtient donc une extension du RCPSP à une ressource ($m = 1$). Chaque tâche possède M_i modes d'exécutions $\mu = 1, \ldots, M_i$, chacun correspondant à une durée $p_{i\mu}$ et une demande en ressource $r_{i\mu}$.

4.3.2.3 Tâches multi-modes et ressources non renouvelables

Un cas plus général considère deux types de ressources : les ressources renouvelables, telles que nous les avons considérées jusqu'à présent, et les ressources non-renouvelables (ou consommables). Une ressource consommable possède un nombre d'unités R_k au début de l'ordonnancement. Dès qu'une tâche demandant r_{ik} unités de cette ressource est démarrée, les r_{ik} unités sont consommées par la tâche et ne seront plus disponibles. Ce type de ressource permet par exemple de représenter un budget où une ressource énergétique.

Dans cette approche, chaque tâche possède M_i modes d'exécutions. Chaque mode $\mu = 1, \ldots, M_i$ correspond à une durée particulière d'exécution $p_{i\mu}$ et à un ensemble de demande sur chaque ressource k (renouvelable ou consommable) $r_{ik\mu}$. Evidemment, globalement, plus la demande en ressource (consommable ou non) est élévée, plus la tâche peut être exécutée rapidement.

La présence de ressources consommables se rapproche des cas pratiques mais augmente considérablement la complexité du problème. En effet, déterminer s'il existe une solution réalisable est en soi très difficile.

4.3.2.4 Disponibilité variable des ressources

La prise en compte des indisponibilités des ressources est souvent indispensable pour appliquer les méthodes d'ordonnancement de projet en pratique. Le modèle et les méthodes présentées dans ce chapitre s'adaptent assez bien à cette caractéristique et remplaçant la disponibilité constante R_k par la disponibilité variable R_{kt}. Notons qu'il reste à définir comment les tâches sont exécutées relativement à ces indisponibilités. On distingue alors les cas où elles ne peuvent être interrompues par une indisponibilité des cas où

elles peuvent être interrompues et reprendre lorsque suffisament d'unités de ressources sont présentes.

On peut également considérer les cas «élastiques» [6] où le nombre d'unités d'une ressource requise par une tâche à un instant donné peut varier, l'énergie totale requise par la tâche sur cette ressource étant constante. La prise en compte très précise des caractéristiques pratiques des ressources et des tâches doit toutefois être maniée avec précaution. Les solutions proposées doivent comporter suffisament de flexibilité pour être adaptée à la réalité du terrain par les décideurs.

4.3.3 Ordonnancement multi-projets

Dans un environnement multi-projets, plusieurs projets différents doivent être exécutés simultanément en partageant un sous-ensemble commun de ressources. Dans ce cas, chaque projet a généralement sa propre date de lancement et sa propre date de livraison. Il est évidemment possible (et souvent préconisé dans la littérature) d'agréger les différents projets en un seul projet pour ce ramener aux méthodes étudiées dans ce chapitre. Pourtant, cette décision dépend du niveau hiérarchique de décision supérieur dans la gestion de projet, qui peut juger opportun de décomposer un gros projet en plusieurs projets relativement indépendants.

Il parait ainsi primordial de tenir compte au contraire de la présence de plusieurs projets. Ceci est mis en évidence dans une étude récente [25], qui propose une comparaison expérimentale de méthodes mono-projet et multi-projets basées sur des règles de priorités et des procédures d'amélioration, dans un environnement multicritères.

Les problèmes de gestion multi-projets dépassent en fait largement le cadre de l'ordonnancement. Nous renvoyons le lecteur par exemple à [21] pour une présentation des nouvelles problématiques de recherche en management de projet.

4.3.4 Gestion des incertitudes : approches stochastiques, robustes et réactives

Les données d'un problème d'ordonnancement de projet ne sont pas toujours connues avec certitude. Pendant la réalisation du projet, les durées des activités peuvent être soumises à des variations, la disponibilité des ressources peut varier de manière imprévue, de nouvelles activités peuvent apparaître, etc.

Deux grand types d'approches existent pour prendre en compte ces indéterminismes. L'ordonnancement de projet stochastique définit les données susceptibles de varier (par exemple les durées des tâches p_i) comme des variables aléatoires. Il est souvent mais pas obligatoirement supposé que les variables aléatoires sont indépendantes et de distribution connue. Les approches stochastiques visent souvent à calculer et à minimiser, avec ou sans contraintes de ressources, l'espérance de la durée totale du projet. Un résultat fondamental justifie l'utilisation de ce type d'approches. On peut en effet prouver (par ex. dans [18]) que l'espérance de la durée totale du projet ne peut être inférieure à la durée totale déterministe du projet en remplaçant les durées des activités par leurs espérances.

Une approche beaucoup plus récente consiste à conserver l'aspect déterministe dans les données mais à prendre en compte les incertitudes de deux manières complémentaires. L'approche robuste consiste à intégrer, dans la solution proposée au problème déterministe, une certaine flexibilité permettant a priori de réagir aux aléas. Bien que controversée, l'approche de la chaîne critique (voir [18] et [20]) est une bonne illustration de ces préoccupations de robustesse. Elle propose d'insérer dans l'ordonnancement des «tampons», périodes d'inactivité susceptibles d'absorber les différentes types d'aléas.

L'approche réactive, complémentaire, permet de réagir a posteriori, une fois un ordonnancement perturbé par un aléa. Il s'agit alors de «réparer» l'ordonnancement perturbé par des méthodes de réordonnancements qui doivent être à la fois rapides et performantes en terme de critères d'évaluation de la solution réparée. Un objectif supplémentaire de stabilité apparaît alors, pour éviter que les solutions successives issues des réparations ne soient trop éloignées.

4.4 Liens avec l'ordonnancement de la production et la génération d'horaires

Nous terminons ce chapitre par la présentation des analogies entre l'ordonnancement de projet sous contraintes de ressources (et ses extensions) et certains des problèmes présentés dans cet ouvrage.

4.4.1 Ordonnancement d'atelier

L'ordonnancement de la production de type *job-shop* (de même que la plupart des problèmes d'ordonnancement d'atelier) sont des cas particuliers du problème de RCPSP. Il s'agit du cas où les ressources sont de disponibilité unitaire $R_k = 1$ et chaque tâche i (correspondant aux *opérations* du job-shop) est telle qu'il existe une et une seule ressource pour laquelle $r_{ik} = 1$. De plus, le graphe des contraintes de précédence décomposables en chaînes indépendante, chaque chaîne constituant un travail.

Dans ce cas, toutes les méthodes basées sur le caractère cumulatif du problème sont alors bien sûr inadaptées.

4.4.2 Génération d'horaires

Nous pouvons illustrer les liens entre le RCPSP et la génération d'horaires en considérant le problème de réalisation des emplois du temps à l'université comme montré par [9]. Le problème est d'affecter m classes à s professeurs pendant T périodes. Chaque professeur j doit donner n_{jl} cours à la classe l, de sorte qu'une classe n'ait au plus qu'un cours à une période donnée, qu'un professeur n'ait au plus qu'un cours à une période donnée en tenant compte des périodes où il est indisponible. Ce problème se représente comme un problème de RCPSP avec $n = \sum_{j=1}^{j=s} \sum_{l=1}^{l=m} n_{jl}$ activités, $s + m$ ressources de capacité unitaire avec des périodes d'indisponibilité pour les m ressources-professeurs.

4.4.3 Un cadre idéal pour l'association de l'ordonnancement de production et la planification de personnel ?

D'un point de vue modélisation, l'ordonnancement de projet permet d'associer les problématiques d'ordonnancement de la production avec les problématiques de la planification de personnel. On peut construire par exemple un modèle où une partie des ressources correspond aux machines de l'atelier et l'autre partie correspond aux équipes de production. Il n'en reste pas moins que les méthodes proposées jusqu'ici pour l'ordonnancement de projet classique peuvent se révéler inadaptées pour résoudre un modèle global par trop complexe et hétérogène.

Bibliographie

[1] Ramòn Alvarez-Valdés et José M. Tamarit, «The project scheduling polyhedron : dimension, facets and lifting theorems», dans *European Journal of Operational Research*, tm. 67, (1993), pp. 204–220.

[2] C. Artigues, P. Michelon et S. Reusser, «Insertion techniques for static and dynamic resource constrained project scheduling», dans *European Journal of Operational Research*, tm. 149, (2003), pp. 249–267.

[3] T. Baar, P. Brucker et S. Knust, «Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem», dans «Meta-heuristics : Advances and Trends in Local Search Paradigms for Optimization», (dirs.) S. Voss, S. Martello, I.Osman et C.Roucairol, Kluwer, 1998, pp. 1–18.

[4] Egon Balas, «Project scheduling with resource constraints», dans «Applications of Mathematical Programming Techniques», (dir.) E.M.L. Beale, American Elsevier, 1970.

[5] Philippe Baptiste et Sophie Demassey, «Tight LP bounds for the resource constrained project scheduling», dans *OR Spectrum*, tm. 26, (2004), pp. 251–262.

[6] Philippe Baptiste, Claude Le Pape et Wim Nuijten, *Constraint-based Scheduling*, no. 39 dans International Series in Operations Research and Management Science, Kluwer Academic Publishers, 2001.

[7] P. Brucker, A. Drexl, R. Möhring, K. Neumann et E. Pesch, «Resource-constrained project scheduling problem : Notation, classification, models and methods», dans *European Journal of Operational Research*, tm. 112, no. 1, (1999), pp. 3–41.

[8] P. Brucker, S. Knust, A. Schoo et O. Thiele, «A branch and bound algorithm for the resource-constrained project scheduling problem», dans *European Journal of Operational Research*, tm. 107, (1998), pp. 272–288.

[9] Peter Brucker, «Complex scheduling problems», Rap. tech. 214, Osnabrücker Schriften zur Mathematik, 1999.

[10] Peter Brucker et Sigrid Knust, «A linear programming and constraint propagation-based lower bound for the RCPSP», dans *European Journal of Operational Research*, tm. 127, (2000), pp. 355–362.

[11] J. Carlier et P. Chrétienne, *Problémes d'ordonnancement*, Masson, 1988.

[12] Jacques Carlier et Bruno Latapie, «Une méthode arborescente pour résoudre les problèmes cumulatifs», dans *RAIRO-Recherche Opérationnelle*, tm. 25, no. 3, (1991), pp. 311–340.

[13] Jacques Carlier et Emmanuel Néron, «A new LP based lower bound for the cumulative scheduling problem», dans *European Journal of Operational Research*, tm. 127, no. 2, (2000), pp. 363–382.

[14] Jacques Carlier et Eric Pinson, «An algorithm for solving the job-shop problem», dans *Management Science*, tm. 35, (1989), pp. 164–176.

[15] Nicos Christofides, Ramòn Alvarez-Valdés et José M. Tamarit, «Project scheduling with resource constraints : a branch and bound approach», dans *European Journal of Operational Research*, tm. 29, no. 3, (1987), pp. 262–273.

[16] Sophie Demassey, Christian Artigues et Philippe Michelon, «Constraint-propagation-based cutting planes : an application to the resource-constrained project-scheduling problem», dans *INFORMS Journal on Computing*, tm. 17, no. 1, (2005), pp. 52–65.

[17] Erik Demeulemeester et Willy Herroelen, «New benchmark results for the resource-constrained project scheduling problem», dans *Management Science*, tm. 43, no. 11, (1997), pp. 1485–1492.

[18] Erik L. Demeulemeester et Willy S. Herroelen, *Project Scheduling : a Research Handbook*, no. 49 dans International Series in Operations Research and Management Science, Kluwer Academic Publishers, 2002.

[19] Michael R. Garey et David S. Johnson, *Computers and Intractibility. A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[20] Vincent Giard, *Gestion de projets*, Economica, 1991.

[21] Vincent Giard, *Faire de la recherche en management de projet*, Vuibert, 2004.

[22] Y. Kochetov et A. Stolyar, «Evolutionary local search with variable neighborhood for the resource-constrained project scheduling problem», dans «3rd International Workshop of Computer Science and Information Technologies», Russia, 2003.

[23] R. Kolisch et S. Hartmann, «Experimental evaluation of heuristics for resource-constrained project scheduling : an update», Rap. tech., Technical University of Munich, 2004.

[24] Rainer Kolisch et S. Hartmann, «Heuristic algorithms for solving the resource-constrained project scheduling problem : Classification and computational analy-

sis», dans «Handbook on Recent Advances in Project Scheduling», (dir.) J. Weglarz, chap. 7, Kluwer Academic Publishers, 1999.

[25] A. Lova, C. Maroto et P. Tormos, «multicriteria heuristic method to improve resource allocation in multiproject scheduling», dans *European Journal of Operational Research*, tm. 127, (2000), pp. 408–424.

[26] A. Mingozzi, V. Maniezzo, S. Ricciardelli et L. Bianco, «An exact algorithm for the multiple resource-constrained project scheduling problem based on a new mathematical formulation», dans *Management Science*, tm. 44, (1998), pp. 714–729.

[27] Rolph H. Möhring, Andreas Schultz, Frederik Stork et Marc Uetz, «Solving project scheduling problems by minimum cut computations», dans *Management Science*, tm. 49, (2003), pp. 330–350.

[28] Emmanuel Néron, *Du flow-shop hybride au problème cumulatif*, Thèse de doctorat, Université Technologique de Compiègne, 1999.

[29] M. Palpant, C. Artigues et P. Michelon, «LSSPER : solving the resource-constrained project scheduling problem with large neighbourhood search», dans *Annals of Operations Research*, tm. 131, (2004), pp. 237–257.

[30] A.A. Pritsker, L.J. Watters et P.M. Wolfe, «Multi-project scheduling with limited resources : a zero-one programming approach», dans *Management Science*, tm. 16, (1969), pp. 93–108.

[31] PSPLIB, «Project scheduling problem library», `http://www.bwl.uni-kiel.de/Prod/psplib`.

[32] B. Roy, *Les problèmes d'ordonnancement - Applications et méthodes*, Dunod, Paris, 1964.

[33] V. Valls, F. Ballestin et M.S. Quintanilla, «A hybrid genetic algorithm for the RCPSP», Rap. tech., Department of Statistics and Operations Research, University of Valencia, 2003.