Chapitre 3

Ordonnancement sous contraintes

Nous introduisons les notions fondamentales en ordonnancement sous contraintes : tâches, contraintes temporelles et contraintes de partage de ressource. Nous évoquons brièvement les techniques de placement sous contraintes apparentées à nos travaux en ordonnancement.

Sommaire

Les problèmes d'ordonnancement et de placement forment une classe de problèmes d'optimisation combinatoire généralement complexe. La programmation par contraintes est dorénavant une approche efficace pour ces problèmes.

Nous rappelons dans ce chapitre les notions d'ordonnancement et placement sous contraintes nécessaires à la compréhension de nos travaux en insistant sur la conception de mécanismes efficaces et généraux de propagation de contraintes qui réduisent l'espace de recherche. Le lecteur est invité à se référer à Lopez et Roubellat [32] ou Baptiste *et al.* [33] pour un état de l'art approfondi. Nous introduisons la notion de tâche (section 3.1) puis deux grandes familles de contraintes d'ordonnancement : les contraintes temporelles (section 3.2) et les contraintes de partage de ressource (section 3.3). Nous présentons ensuite le modèle disjonctif (section 3.4) qui sera discuté dans nos travaux sur les problèmes d'atelier. Nous ne traiterons pas des stratégies de recherche dans ce chapitre. Elles seront introduites au cas par cas dans la deuxième partie de la thèse.

Finalement, nous rappelons quelques résultats de placement sous contraintes (section 3.5). En effet, nous utilisons une contrainte globale de placement à une dimension dans nos travaux sur les problèmes de fournées. Par ailleurs, nous insisterons sur la parenté entre les problèmes d'ordonnancement et certains problèmes de placement à deux dimensions.

3.1 Tâches

Une tâche ou activité $i \in [1, n]$, notée T_i, est une entité élémentaire de travail localisée dans le temps par une date de début s_i (*start*) et de fin e_i (*end*), dont la réalisation est caractérisée par une durée positive p_i (*processing time*). La fonction booléenne $\delta_i(t)$ est la fonction de Dirac de l'ensemble des moments $\mathcal{M}_i$ où la tâche T_i est exécutée :

$$s_i = \min(\mathcal{M}_i) \qquad e_i = \max(\mathcal{M}_i) \qquad p_i = \text{card}(\mathcal{M}_i)$$

Une tâche est dite préemptive si elle peut être morcelée afin de diminuer l'inactivité des ressources ($s_i + p_i \leq e_i$). Dans les problèmes non préemptifs, les tâches ne peuvent pas être interrompues ($s_i + p_i = e_i$ et $\mathcal{M}_i = [s_i, e_i[$).
Nous définissons quelques notations relatives à l'exécution d'une tâche T_i en suivant la terminologie anglaise de la littérature illustrée en figure 3.1. Les notations $est_i = \min(s_i)$ et $lst_i = \max(s_i)$ désigneront respectivement les dates de début au plus tôt (*earliest starting time*) et au plus tard (*latest starting time*) d'une tâche T_i alors que $ect_i = \min(e_i)$ et $lct_i = \max(e_i)$ désigneront ses dates de fin au plus tôt (*earliest completion time*) et au plus tard (*latest completion time*). La fenêtre de temps d'une tâche (*time window*) désigne l'intervalle $[est_i, lct_i[$ durant lequel la tâche est potentiellement exécutée. La partie obligatoire (*compulsory part*) [34] d'une tâche désigne l'intervalle $[lst_i, ect_i[$ durant lequel la tâche est obligatoirement exécutée (représenté par un rectangle dans la figure 3.1). Notez que la partie obligatoire d'une tâche peut être vide.

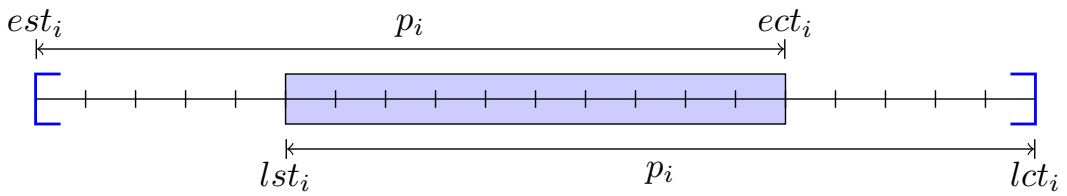


FIGURE 3.1 – Notations relatives à l'exécution d'une tâche.

Deux tâches fictives de durées nulles, T_{start} et T_{end}, représentant le début et la fin de l'ordonnancement sont généralement ajoutées. Elles sont généralement reliées aux autres tâches par les contraintes arithmétiques $s_i - s_{\text{start}} \geq 0$ et $e_{\text{end}} - e_i \geq 0$. Sans perte de généralité, nous supposons dorénavant que le début de l'ordonnancement est fixé à l'origine des temps ($s_{\text{start}} = 0$) et que la fin du projet est égale à son délai total ($e_{\text{end}} = \max_I(e_i)$).

Pour une tâche non préemptive, ou non morcelable, l'ensemble des moments d'exécution $\mathcal{M}_i$ est un intervalle. Une unique variable s_i ou e_i suffit à caractériser une tâche non préemptive de durée fixe. Un couple de variables (s_i, p_i), (s_i, e_i) ou (p_i, e_i) suffit à caractériser une tâche non préemptive de durée variable, c'est-à-dire qui n'est pas connue à l'avance. Néanmoins, `choco` utilise un triplet de variables entières positives (s_i, p_i, e_i) sous une contrainte de non-préemption $s_i + p_i = e_i$. La raison principale est la granularité plus fine offerte par cette représentation pour les tâches à durée variable. Par exemple, une partie de l'information sur la tâche ($s_i \in [0, 10], p_i \in [1, 5], e_i \in [1, 11]$) est nécessairement perdue lorsqu'elle est seulement représentée par un couple de variables. Ce choix est aussi guidé par des raisons plus spécifiques de `choco` détaillées dans le chapitre 9. Les expérimentations suggèrent que ce choix a une influence acceptable sur les performances du solveur lorsque toutes les tâches ont des durées fixes.

L'objectif des problèmes d'ordonnancement est de minimiser une mesure de performance. Certaines dépendent exclusivement des dates de fin des tâches, notée C_i selon les conventions de recherche opérationnelle. Le délai total ou date d'achèvement maximale d'un ordonnancement C_{max} (*makespan*) est la date de fin de la dernière tâche à quitter le système. Une valeur minimale de C_{max} correspond généralement à une utilisation intensive des ressources. La délai moyen, pondérée $\sum w_i C_i$ ou non $\sum C_i$ peut aussi indiquer le coût de l'ordonnancement.

3.2 Contraintes temporelles

Cette section traite de la représentation des contraintes temporelles en ordonnancement sous contraintes. Elles intègrent les contraintes de temps alloué issues généralement d'impératifs de gestion et relatives aux dates limites des tâches (disponibilité des approvisionnements, délai de livraison) ou au délai total d'un projet, mais aussi les contraintes d'enchaînement qui définissent le positionnement relatif de certaines tâches par rapport à d'autres.
Ces contraintes peuvent toutes s'exprimer à l'aide d'*inégalités de potentiels* [35] qui imposent une distance minimale d_{ij} entre deux instants particuliers associés aux tâches (le plus souvent les dates de début) : $x_j - x_i \geq d_{ij}$. Ainsi, le réseau de contraintes est souvent représenté par un graphe de distance $\mathcal{G} = (\mathcal{N}, V)$ dans lequel les nœuds représentent les variables et les arcs (i, j) de valuation d_{ij} représentent les contraintes temporelles. Les deux tâches fictives T_{start} et T_{end} sont généralement ajoutées à ce graphe par les contraintes arithmétiques présentées précédemment.
Dans cette thèse, nous n'utiliserons que les contraintes temporelles introduites dans cette section.

3.2.1 Contraintes de précédence

Une contrainte de précédence entre deux tâches T_i et T_j, notée symboliquement $T_i \prec T_j$ (T_i précède T_j), est représentable par une unique inégalité de potentiel (3.1). Il s'agit d'un cas particulier de la contrainte de précédence avec temps d'attente $d_{ij} \geq 0$ (3.2). Le temps d'attente représente, par exemple, le temps d'entretien d'une machine, le temps de refroidissement ou de séchage d'une pièce, etc.

$$T_i \preceq T_j \quad \Leftrightarrow \quad s_j - e_i \geq 0 \tag{3.1}$$
$$T_i \prec T_j \quad \Leftrightarrow \quad s_j - e_i \geq d_{ij} \tag{3.2}$$

Le filtrage d'une précédence consiste généralement à appliquer une règle d'arc-consistance pour l'inégalité de potentiel sur les bornes des variables.

3.2.2 Contraintes de disponibilité et d'échéance

Les contraintes de dates limites d'une tâche T_i peuvent également s'exprimer à l'aide d'inégalités de potentiel entre T_i, T_{start} et T_{end}. La contrainte de disponibilité $s_i \geq r_i$ se réécrit sous la forme $s_i - s_{\text{start}} \geq r_i$ et la contrainte d'échéance $e_i \leq d_i$ devient $s_{\text{start}} - e_i \geq -d_i$. En général, on réduit directement les fenêtres de temps (via les domaines) lorsque les dates limites sont connues à l'avance.

3.2.3 Contraintes de disjonction

Une expression peut relier plusieurs inégalités de potentiel par des connecteurs logiques. Une disjonction d'inégalités de potentiel est une expression dans laquelle apparaît le connecteur logique $\vee$ (ou logique). Sur le plan sémantique, la contrainte est satisfaite si au moins un des littéraux est une contrainte satisfaite. Une contrainte de disjonction, ou non-chevauchement, entre deux tâches T_i et T_j est satisfaite si les tâches s'exécutent dans des fenêtres de temps disjointes (3.3). Il s'agit à nouveau d'un cas particulier de la contrainte de disjonction avec temps d'attente (3.4). Cette contrainte peut par exemple exprimer l'existence de deux gammes opératoires différentes. Arbitrer une disjonction consiste à déterminer l'ordre relatif entre les tâches, c'est-à-dire choisir quelle précédence (littéral) est satisfaite.

$$T_i \simeq T_j \quad \Leftrightarrow \quad (T_i \preceq T_j) \vee (T_j \preceq T_i) \tag{3.3}$$
$$T_i \sim T_j \quad \Leftrightarrow \quad (T_i \prec T_j) \vee (T_j \prec T_i) \tag{3.4}$$

Tant que la disjonction n'est pas arbitrée, le filtrage applique deux règles de séquencement : Précédence Interdite (PI) - Précédence Obligatoire (PO). Une précédence $T_i \prec T_j$ est dite interdite lorsque les fenêtres de temps des tâches sont incompatibles avec la décision, ce qui rend l'autre précédence obligatoire. Une précédence $T_i \prec T_j$ est dite obligatoire dès qu'elle est directement impliquée par les fenêtres de temps. La règle (PI) implique (PO), mais elles réagissent à différents types de modification des domaines. Ces

règles détectent une inconsistance lorsqu'aucun séquencement n'est possible.

$$ect_i + d_{ij} \geq lst_j \quad \Rightarrow \quad T_j \prec T_i \tag{PI}$$
$$lct_i + d_{ij} \leq est_j \quad \Rightarrow \quad T_i \prec T_j \tag{PO}$$

3.2.4 Problèmes temporels

Les problèmes temporels consistent à ordonnancer un ensemble de tâches liées uniquement par des contraintes temporelles [36]. Le *problème d'ordonnancement de projet* connu sous l'acronyme PERT consiste à ordonnancer un ensemble de tâches liées par des contraintes de précédence sans limitation de ressources. Le *problème central de l'ordonnancement* consiste à ordonnancer un ensemble des tâches liées par des contraintes de précédences généralisées (le temps d'attente peut être négatif). Nous supposons que $d_{ij} \geq -d_{ji}$ pour éviter que le problème ne soit trivialement insatisfiable. Le graphe de distance permet de résoudre ce problème grâce à sa transformation en un problème (polynomial) de flot ou de plus court chemin [37]. De plus, il a été montré que le problème est consistant en l'absence de cycle de coût négatif et peut, dans ce cas, être résolu sans backtrack. Il est donc possible de calculer les ordonnancements au plus tôt et au plus tard en temps polynomial et même linéaire dans le cas du problème d'ordonnancement de projet. Les ordonnancements au plus tôt et au plus tard peuvent alors servir à la réduction de la fenêtre des temps des tâches même en présence de limitation de ressources. Nous reviendrons sur la résolution du problème d'ordonnancement de projet avec le solveur Choco en Annexe B. Quand on ne peut pas résoudre le problème sans backtrack, certains travaux portent sur la détection de cycle et la propagation incrémentale lors de l'ajout ou du retrait dynamique de contraintes temporelles.

3.3 Contraintes de partage de ressource

Une ressource r est un moyen technique ou humain requis pour la réalisation d'une tâche et disponible en quantité limitée, sa capacité entière positive C_r (supposée constante). Plusieurs types de ressource sont à distinguer. Une ressource est *renouvelable* si après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible en même quantité (les humains, les machines, l'espace ...). Dans le cas contraire, elle est *consommable* (matières premières, budget ...), c'est-à-dire la consommation globale au cours du temps est limitée. Une ressource peut aussi être *doublement contrainte* lorsque son utilisation instantanée et sa consommation globale sont toutes deux limitées (source d'énergie, financement ...). D'autres types de ressource (producteur – consommateur, taux de renouvellement ...) définis en fonction de l'évolution de la capacité instantanée ou globale lors de l'exécution des tâches (produits manufacturés ou naturels ...) ne seront pas discutés dans cette thèse. On distingue par ailleurs les ressources disjonctives ($C_r = 1$) qui ne peuvent exécuter qu'une tâche à la fois, des ressources cumulatives ($C_r \geq 1$) qui peuvent être utilisées par plusieurs tâches simultanément.
La consommation d'une tâche T_i sur une ressource renouvelable r est caractérisée par une consommation totale d'énergie $w_i(r) \geq 0$, une consommation minimale à chaque instant ou hauteur $h_i(r)$ et une consommation instantanée $w_i(r,t)$ à l'instant t. Dans le cas *fully elastic* (FE), la capacité de la ressource doit être respectée à chaque instant, mais il n'y a aucune restriction sur l'utilisation de la ressource par les tâches. La contrainte (3.5) impose que l'énergie consommée par une tâche lors de son exécution soit supérieure à sa consommation minimale requise. La contrainte (3.7) limite l'énergie consommée par une tâche à chaque instant. La contrainte (3.6) assure que l'énergie totale consommée par une tâche sur l'horizon de planification $[0, h]$ est égale à sa consommation totale. Finalement, la contrainte (3.8) limite l'énergie totale consommée par les tâches à chaque instant. Les contraintes supplémentaires $w_i(r,t) = \delta_i(t)$ et $w_i(r,t) = h_i(r) \times \delta_i(t)$ sont respectivement ajoutées pour représenter la consommation constante des tâches sur les ressources disjonctives et cumulatives. Soit $u_i(r)$ une variable booléenne indiquant si la tâche i utilise la ressource r, le modèle obtenu en substituant l'expression non linéaire $\delta_i(t) \times u_i(r)$ à la variable $\delta_i(t)$ est valable pour les problèmes d'allocation de ressources.

$$w_i(r,t) \geq h_i(r) \times \delta_i(t) \quad (3.5) \qquad w_i(r,t) \leq C_r \times \delta_i(t) \quad (3.7)$$

$$\sum_{t=0}^{h} w_i(r,t) = w_i(r) \quad (3.6) \qquad \sum_{i=1}^{n} w_i(r,t) \leq C_r \quad (3.8)$$

Cette représentation induit une hiérarchie entre les ressources illustrée par le diagramme de la figure 3.2 dans lequel la relation A → B indique que le type de ressource B est un cas particulier du type A. Le cas

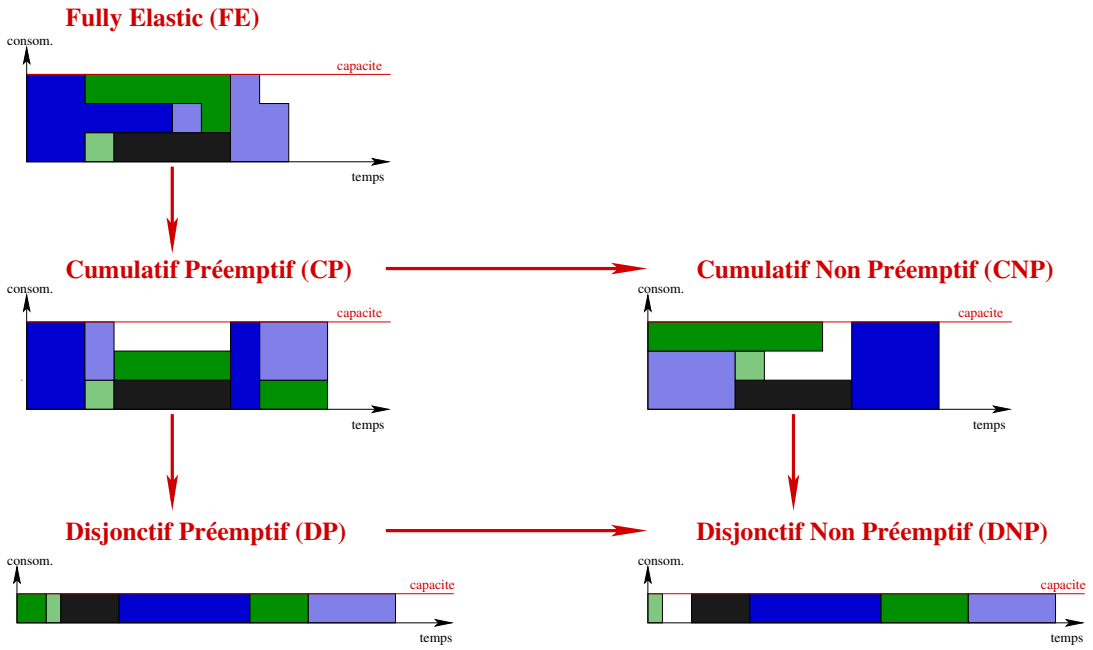


FIGURE 3.2 – Hiérarchie des types de ressources.

général *fully elastic* est intéressant puisque ses propriétés et règles de filtrage sont valides pour tous les autres types de ressource. Les techniques de *time-tabling* permettent de réaliser le filtrage des ressources en maintenant l'arc-consistance sur les contraintes (3.8) puis l'ajustement des dates de début et de fin des tâches en fonction de leur consommation instantanée d'énergie $w_i(r,t)$. Elles sont peu efficaces dans le cas général, souvent appliquées dans le cas préemptif, et spécialisées dans le cas non préemptif. Nous présentons maintenant les principes du filtrage des contraintes disjonctive et cumulative dans le cas non préemptif.

3.3.1 Contrainte disjonctive

La prise en compte d'une contrainte disjonctive définit un problème de séquencement dans lequel il faut ordonner totalement les tâches qui utilisent la ressource. La prise en compte des fenêtres de temps réduit l'ensemble des solutions : certaines configurations sont interdites ; d'autres deviennent obligatoires.

Le concept d'énergie permet d'effectuer des raisonnements quantitatifs intégrant les contraintes de temps et de ressource. Ces raisonnements font appel à un bilan de l'utilisation d'une ressource sur un nombre quadratique d'intervalles temporels pertinents [38] pour l'identification et le calcul de différentes énergies. Sur un intervalle de temps, l'énergie est fournie par la ressource et consommée par les tâches. La règle d'*overload checking* (OC) consiste à lever une contradiction lorsque le bilan énergétique d'un intervalle temporel est déficitaire, c'est-à-dire la consommation d'énergie minimale des tâches sur cet intervalle dépasse l'énergie totale disponible. Ce type de raisonnement permet d'interdire la localisation des tâches dans certains intervalles temporels qui engendrerait un bilan énergétique déficitaire.

D'autres règles de filtrage dominant les raisonnements énergétiques dans le cas disjonctif déduisent des conditions de séquencement des tâches qui sont ensuite propagées et peuvent entraîner un ajustement [39]

des fenêtres d'exécution des tâches.
La règle *not first/not last* (NF/NL) généralise la règle (PI) en déduisant qu'une tâche T_i ne peut être ordonnancée avant/après un sous-ensemble Ω d'autres tâches partageant la ressource disjonctive. Cette règle entraîne un ajustement de la fenêtre de temps, car il existe au moins une tâche de Ω placée avant/après la tâche T_i dans un ordonnancement admissible. Baptiste et Le Pape [38], Torres et Lopez [40] ont proposé des algorithmes pour cette règle dont la complexité est quadratique.
Réciproquement, la règle d'*edge finding* (EF) détermine qu'une tâche T_i est obligatoirement ordonnancée avant/après un sous-ensemble de tâches Ω. La fenêtre de temps de la tâche est alors ajustée en estimant la date de fin au plus tôt ou de début au plus tard des tâches de l'ensemble Ω. Caseau et Laburthe [41, 42] ont proposé un algorithme en $O(n^3)$ basé sur les intervalles de tâches. Ils maintiennent ces intervalles de tâches incrémentalement durant la recherche et utilisent un système de règle d'activation des intervalles pour améliorer l'efficacité de leur méthode. Des déductions supplémentaires peuvent être réalisées grâce aux intervalles de tâches. Carlier et Pinson [39] ont proposé le premier algorithme en $O(n \log n)$ n'utilisant pas les intervalles de tâches mais une structures de données plus complexe.

Vilím [43], Vilím *et al.* [44] ont récemment proposé une implémentation des règles précédentes avec une complexité en $O(n \log n)$. Ils utilisent une structure d'arbre binaire équilibré qui leur permet d'abaisser la complexité et de proposer une nouvelle règle de filtrage : *detectable precedence* (DP). Cette règle propage les précédences découvertes en inspectant les fenêtres de temps des tâches. La fenêtre de temps de chaque tâche est ajustée en fonction de l'ensemble de ses prédécesseurs ou successeurs. Cette règle s'applique quelquefois lorsque les règles précédentes ne permettent aucun ajustement. En effet, l'*edge finding* peut identifier que T_i précède T_k et que T_j précède T_k ($T_i \preceq T_k$ et $T_j \preceq T_k$). Il peut arriver qu'aucune de ces précédences ne permette d'ajustement des bornes. L'idée consiste alors à exploiter l'information $\{T_i, T_j\} \preceq T_k$.
Tous les algorithmes reposent sur le calcul de la date de fin au plus tôt ECT_Ω (*earliest completion time*) d'un ensemble de tâches Ω partageant une ressource disjonctive :

$$ECT_\Omega = \max \left\{ \min_{T_i \in \Omega} \{est_i\} + \sum_{T_i \in \Omega'} p_i,\ \Omega' \subseteq \Omega \right\} \tag{3.9}$$

Vilím [43] a proposé des structures de données efficaces : les arbres Θ et Θ-Λ. Ils sont basés sur une structure d'arbre binaire équilibré permettant un calcul incrémental de ECT_Ω avec une complexité de $O(\log(n))$ lors de l'ajout ou du retrait d'une tâche de l'ensemble Ω.

Van Hentenryck *et al.* [45] utilisent la construction de disjonction pour supprimer les valeurs inconsistantes dans les deux séquencements possibles d'une paire de tâches. Plus récemment, Wolf [46] ajuste les fenêtres de temps des tâches en fonction de leur nombre maximal de prédécesseurs et de successeurs. Hooker [47] a proposé différentes relaxations linéaires. La règle de Jackson [48] construit un ordonnancement préemptif qui est une condition nécessaire pour la contrainte disjonctive.

De nombreuses extensions de la contrainte ont été proposées. Par exemple, une version *soft* de la contrainte, c'est-à-dire une relaxation, a été proposée pour maximiser le nombre de tâches exécutées par la ressource sous des contraintes de dates limites [49]. Ce principe a été étendu depuis grâce à la notion de *ressource alternative* pour laquelle certaines tâches sont optionnelles, c'est-à-dire qu'elles peuvent ne pas être exécutées par la ressource [44, 50–52]. Des travaux préliminaires sur un problème d'allocation de ressource seront discutés en Annexe D. Les algorithmes de filtrage dépendent de l'appartenance des tâches non exécutées à l'ordonnancement final ou non.
Barták et Cepek [53] maintiennent aussi la fermeture transitive des précédences concernant les tâches de la ressource pour déduire de nouvelles précédences et ajuster les fenêtres de temps.

3.3.2 Contrainte cumulative

La contrainte cumulative impose que la somme des hauteurs des tâches s'exécutant à un instant donné soit inférieure à la capacité de la ressource. La contrainte peut aussi imposer un niveau de consommation minimale aux instants où des tâches s'exécutent sur la ressource.

Les premiers algorithmes de filtrage sont basés sur la construction d'un profil cumulatif qui agrège les parties obligatoires des tâches afin d'ajuster leurs fenêtres de temps pour éviter un dépassement de la capacité de la ressource. Même si les tâches avec une partie obligatoire vide sont ignorées, l'algorithme à

balayage (*sweep*) proposé par Beldiceanu et Carlsson [54] gèrent des cas généraux (ressources alternatives, producteur – consommateur ...) tout en permettant un passage à l'échelle grâce à une complexité réduite de $O(n \log n)$. Nous rappelons brièvement le principe des algorithmes à balayage : l'algorithme déplace une ligne verticale (*sweep-line*) et utilise principalement deux structures de données :
Sweep-line status : cette structure contient les informations sur la position courante Δ de la *sweep-line*.
Event-point series : cette structure contient les événements à examiner triés en ordre croissant.
L'algorithme initialise la *sweep-line status* à la valeur initiale de Δ. La *sweep-line* parcours ensuite les événements en mettant à jour la *sweep-line status*. Dans notre cas, la *sweep-line* parcourt les valeurs du domaine d'une variable T_i et la *sweep-line status* contient un ensemble de contraintes qui doivent être vérifiées à l'instant Δ. Si pour un instant Δ, l'ensemble des contraintes est insatisfiable, alors on peut enlever la valeur Δ du domaine de T_i.
Une propriété intéressante de cet algorithme est que l'on ne considère qu'un nombre linéaire d'évènements à chaque appel.

Les algorithmes basés sur les raisonnements énergétiques considèrent la consommation d'ensemble de tâches sur certains intervalles temporels sans se restreindre aux parties obligatoires [55]. Ainsi, les règles d'*edge finding* [56] ont été adaptées au cas cumulatif ainsi que l'arbre-Θ-Λ devenu arbre-Φ [57].

Une condition nécessaire pour la contrainte cumulative est obtenue en imposant une contrainte disjonctive sur l'ensemble des tâches tel que la somme des hauteurs de chaque paire de tâches est supérieure à la capacité. Il existe des techniques plus sophistiquées pour extraire les cliques de disjonctions lorsque plusieurs ressources disjonctives et cumulatives sont impliquées [58].

De nombreuses extensions existent : la cumulative colorée où la consommation est égale au nombre de couleurs distinctes associées aux tâches s'exécutant à un instant donné ; la cumulative avec niveau de priorité où une capacité relative à un niveau donné s'applique à l'ensemble des tâches de priorité inférieure à ce même niveau ; les variantes avec des tâches optionnelles.

3.4 Modèle disjonctif

Le modèle disjonctif [59] est un graphe permettant de décrire les contraintes de précédence et de disjonction dans un problème d'ordonnancement utilisé aussi bien par les méthodes approchées que les méthodes exactes, notamment dans les chapitres 6 et 7. Plus précisément, pour un problème d'ordonnancement de n tâches, le *graphe disjonctif généralisé* $\mathcal{G} = (\mathcal{T}, \mathcal{P}, \mathcal{D}, \mathcal{L})$ est défini comme suit :

- $\mathcal{T}$ est l'ensemble des *nœuds*, représentant les tâches ainsi que deux tâches fictives T_{start} et T_{end} ;
- $\mathcal{P}$ est l'ensemble des *arcs* (orientés) représentant les contraintes d'enchaînement ;
- $\mathcal{D}$ est l'ensemble des *arêtes* (non orientées) représentant les disjonctions ;
- $\mathcal{L}$ est l'ensemble des *valuations* positives donnant pour chaque arc (T_i, T_j) le temps d'attente entre la fin de la tâche T_i et le début de la tâche T_j.

Nous l'appellerons simplement *graphe disjonctif* lorsque les valuations $\mathcal{L}$ sont nulles. Le *graphe de précédence* le sous-graphe orienté $(\mathcal{T}, \mathcal{P})$.

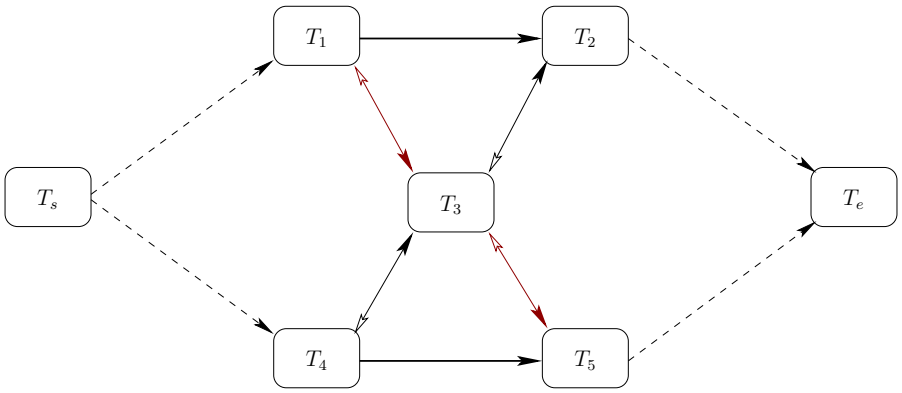
Arbitrer une disjonction revient à transformer une arête $T_i \sim T_j$ en un arc $T_i \prec T_j$ ou $T_j \prec T_i$. La contrainte de non-préemption $s_i + p_i = e_i$ doit aussi être satisfaite en chaque nœud.
L'existence d'un cycle entraîne une inconsistance triviale du problème. De plus, tout ordonnancement réalisable correspond à un arbitrage complet de $\mathcal{G}$, dans lequel toutes les disjonctions sont arbitrées sans créer de circuit. Lorsque toutes les contraintes portant sur les tâches sont représentées dans $\mathcal{G}$, la consistance d'un arbitrage complet implique l'existence d'un ordonnancement dont le délai total correspond à la longueur d'un plus long chemin de $\mathcal{G}$ appelé *chemin critique*. Les activités de ces chemins (il peut en exister plusieurs), dites activités critiques, déterminent à elles seules la date de fin du projet C_{max}. Il est nécessaire de réduire la longueur des chemins critiques pour réduire C_{max}. La marge (*slack time*) est le temps pendant lequel une tâche peut être retardée sans retarder le projet. Le temps de latence des activités critiques est donc nul alors que celui des activités non critiques est strictement positif. Pour un graphe de k arcs, ce chemin est calculable en $O(k)$ par une variante de l'algorithme de Bellman [37]. Beaucoup de métaheuristiques et de recherches arborescentes explorent les arbitrages complets pour en trouver un de délai total minimum. Dans une recherche arborescente, une *heuristique de sélection de disjonction* sélectionne la prochaine disjonction à arbitrer, alors qu'une *heuristique de sélection d'arbitrage* détermine la première précédence à essayer.

T_i	1	2	3	4	5
p_i	6	7	5	11	2
est_i	0	16	11	0	16
lst_i	5	16	11	21	16

(a) Durées d'exécution

$$T_1 \preceq T_2 \quad T_4 \preceq T_5$$
$$T_1 \simeq T_3 \quad T_2 \simeq T_3 \quad T_4 \simeq T_3 \quad T_5 \simeq T_3$$

(b) Contraintes temporelles



(c) Arbitrage d'un graphe disjonctif

FIGURE 3.3 – Construction et d'arbitrage d'un graphe disjonctif.

La construction et l'arbitrage complet d'un graphe disjonctif sont illustrés dans la figure 3.3. Nous cherchons à minimiser le délai total de l'ordonnancement des tâches définies dans la figure 3.3(a) sous les contraintes temporelles en figure 3.3(b). L'arbitrage du graphe disjonctif de la figure 3.3(c) représente des ordonnancements optimaux de délai total égal à 23. Un sommet du graphe disjonctif est associé à chaque tâche et les arcs orientés représentent les précédences alors que les arcs bidirectionnels représentent les disjonctions dont les arbitrages sont indiqués par la direction de la flèche pleine. Le chemin $(T_{\text{start}}, T_4, T_3, T_2, T_{\text{end}})$ de longueur $11+5+7=23$ est l'unique chemin critique. Les valeurs est_i et lst_i de la figure 3.3(a) correspondent aux ordonnancements au plus tôt et au plus tard de cet arbitrage complet.

3.5 Placement sous contraintes

Les problèmes de placement consistent à ranger des articles caractérisés par leur forme(s) dans une ou plusieurs boîtes. Les variantes se distinguent en fonction de la dimension, de la connaissance a priori des articles (*on-line*, ou *off-line*), de la forme des articles et des boîtes (carré, rectangulaire, circulaire ...), de la possibilité de modifier l'orientation des articles. On distingue aussi le problème de faisabilité, c'est-à-dire, existe-t-il un rangement réalisable des articles dans les boîtes, des problèmes d'optimisation, par exemple la minimisation du nombre de boîtes (*bin packing*) ou la minimisation des dimensions d'une seule boîte (hauteur – *strip packing*, aire – *rectangle packing*, volume ...), ou encore la maximisation de la valeur du rangement (problèmes de sac à dos – *knapsack problems*). Le lecteur peut se référer à la bibliographie annotée de [60] ou à une typologie précise des problèmes de placement et de découpe [61, 62]. Dans cette section, nous nous intéresserons particulièrement aux problèmes de placement à une ou deux dimensions qui partagent certaines propriétés avec les problèmes d'ordonnancement.

3.5.1 Placement en une dimension

Nous considérons dans cette section un ensemble d'articles I caractérisés par leur longueur entière positive s_i et un ensemble de conteneurs caractérisés par leur capacité C ($s_i \leq C$).
Le problème *subset sum* n'est pas à proprement parler un problème de placement ou d'ordonnancement, mais il apparaît parfois comme un sous-problème pour obtenir des évaluations par défaut lors du filtrage. Il consiste à maximiser l'espace occupé par le rangement d'un sous-ensemble d'articles dans une boîte. Ce problème est NP-complet malgré son apparente simplicité. Toutefois, il est résolu très efficacement

par la programmation dynamique avec une complexité pseudo linéaire $O(\max(s_i)n)$ [63].
Le problème de *bin packing* à une dimension consiste à minimiser le nombre de boîtes nécessaires pour ranger un ensemble d'articles. De nombreux travaux traitent de la qualité des solutions fournies par des algorithmes polynomiaux d'approximation. Par exemple, les heuristiques de permutation *first fit decreasing* ou *best fit decreasing* rangent les articles par ordre décroissant de taille en les plaçant respectivement dans le premier conteneur disponible ou dans un conteneur disponible dont l'espace restant est minimal. Une permutation des articles définit une classe de rangement.
La recherche arborescente en profondeur de Martello et Toth [64] considère en chaque nœud le rangement du plus grand article restant dans une boîte partiellement remplie ou vide. Chaque nœud est évalué par le calcul d'une borne inférieure dédiée et de la meilleure borne supérieure fournie par des heuristiques. Depuis, diverses approches exactes ont été proposées basées sur des formulations en nombres entiers couplées avec des relaxations linéaires, notamment par la génération de colonnes [65], des méthodes hybrides [66] ou la programmation par contraintes [67–72]. Korf [69], Fukunaga et Korf [71] utilisent un schéma de séparation *bin completion* différent de celui de Martello et Toth [64] qui consiste à ajouter une boîte réalisable au rangement à chaque nœud. Une boite réalisable contient un sous-ensemble des articles libres respectant la capacité de la boîte. Toutes les méthodes mentionnées ci-dessus utilisent intensivement des règles d'équivalence et de dominance ainsi que des bornes inférieures et supérieures dynamiques pour réduire l'espace de recherche. Par exemple, l'efficacité de la *bin completion* dépend exclusivement des règles de dominances entre les boîtes réalisables.
Au contraire, Shaw [70] a proposé une contrainte globale dont le filtrage associé au modèle de Martello et Toth [64] utilisent des raisonnements de type sac à dos et une borne inférieure sur le nombre de boîtes exploitant l'affectation partielle courante indépendamment de toute symétrie ou dominance. La signature de la contrainte contient deux ensembles de variables représentant respectivement la boîte où un article est rangé et l'ensemble des articles rangés dans une boîte. La seule restriction concerne la longueur des articles qui doit être connue à l'avance, mais les autres variables ont des domaines arbitraires. Ainsi, sa conception favorise son intégration dans des modèles complexes avec des contraintes additionnelles (incompatibilités, précédences) ou des objectifs différents tout en restant efficace sur les purs problèmes de *bin packing*. Par exemple, Schaus et Deville [72] ont complété le filtrage pour la gestion de contraintes de précédences dans des problèmes de chaîne d'assemblage. Cependant, le filtrage des rangements réalisables dans une seule boîte n'atteint pas la consistance d'arc généralisée au contraire de celui proposé par [73] pour les problèmes de sac à dos. Nous utiliserons cette contrainte globale dans le modèle pour le problème d'ordonnancement d'une machine à traitement par fournées au chapitre 8.

3.5.2 Placement en deux dimensions

Nous considérons un ensemble d'articles I de forme rectangulaire caractérisés par leur largeur w_i et leur hauteur h_i et de conteneurs caractérisés par leur largeur W et leur hauteur H. Le rangement des articles dans un conteneur respecte les trois contraintes suivantes : (a) chaque article est entièrement inclus dans le conteneur ; (b) Les articles ne se chevauchent pas ; (c) Le rangement est orthogonal, c'est-à-dire le bord des articles est parallèle au bord du conteneur. Dans la plupart des travaux mentionnés, l'orientation des articles est fixée. Nous nous restreindrons aux approches en programmation par contraintes. Le lecteur intéressé pourra se référer à la classification proposée par Dyckhoff [61] et à l'état de l'art de Lodi *et al.* [74] sur les modèles mathématiques, les bornes inférieures, les méthodes d'approximation et les heuristiques et métaheuristiques.

Plusieurs modèles coexistent pour représenter une solution au problème de faisabilité qui consiste à trouver un rangement admissible des articles dans une boîte. Dans le plus intuitif, une solution est un vecteur contenant les coordonnées de chaque article. Fekete et Schepers [75], Fekete *et al.* [76] ont proposé un modèle basé sur un graphe d'intervalle pour chaque dimension (un graphe d'intervalle est le graphe d'intersection d'un ensemble d'intervalles). Ceux-ci correspondent à une *classe de rangement*, c'est-à-dire un ensemble de rangements partageant certaines propriétés. Cette représentation évite l'énumération d'un grand nombre de rangements symétriques appartenant à la même classe. Les algorithmes pour reconnaître une classe de rangement réalisable et reconstruire une solution sont polynomiaux. Moffitt et Pollack [77] utilisent le concept de *meta-CSP* dans lequel deux graphes représentent les disjonctions sur les deux dimensions. Ce concept peut être vu comme une extension du modèle disjonctif dans le cas à

deux dimensions. La différence principale réside dans le fait que l'arbitrage d'une disjonction entre deux articles est réalisé sur l'une ou l'autre dimension. Korf [78] utilise simplement une matrice booléenne binaire de la taille du conteneur dans laquelle les cellules de la matrice représentent une coordonnée dont la valeur est 1 si elle est occupée par un rectangle. Le dernier modèle est surtout répandu parmi les heuristiques et métaheuristiques, mais il est intéressant de par sa simplicité : une permutation des articles définit une classe de rangement. Par exemple, l'heuristique *bottom-left* [79] consiste à ranger à chaque itération le prochain article de la liste à la position le plus en bas et à gauche de la boîte.

Moffitt et Pollack [77] cherchent à trouver un conteneur d'aire minimale contenant tous les articles grâce au *meta-CSP*. On peut remarquer le lien entre cet objectif et la minimisation du délai total dans le modèle disjonctif. Cependant, son approche peut être facilement adaptée au problème de faisabilité. Il emploie différentes techniques pour réduire son espace de recherche, notamment le *forward-checking*, l'utilisation de la transitivité pour éliminer des variables, ainsi que le *semantic branching* qui consiste à utiliser l'espace déjà exploré pour rajouter des *coupes*. Sa stratégie de branchement détecte et élimine certaines symétries des placements, et utilise des heuristiques de sélection de variable et de valeur.
Korf [78, 80] traite le même problème par l'application de règles de dominance entre affectations partielles et d'une borne inférieure sur l'aire perdue dans le conteneur sur une représentation matricielle.
D'autres approches sont basées sur l'ordonnancement cumulatif. Le problème d"ordonnancement cumulatif est une relaxation du problème de faisabilité consistant à relâcher la contrainte d'intégrité des rectangles. La satisfiabilité du problème cumulatif associé à chaque dimension d'un problème de faisabilité est même une condition nécessaire à la satisfiabilité du problème de faisabilité mais pas une condition suffisante. Nous illustrons ce point sur un exemple issu de [12] présenté en figure 3.4.

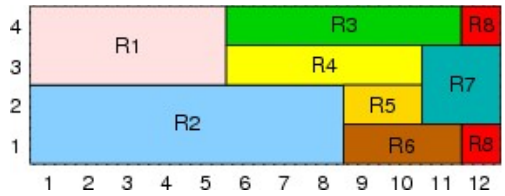


FIGURE 3.4 – Une condition nécessaire mais pas suffisante pour le problème de faisabilité.

Clautiaux *et al.* [81] utilisent une contrainte cumulative sur chaque dimension et vérifie les contraintes de non-chevauchement grâce à un modèle basique. Le filtrage est renforcé par (a) des raisonnements énergétiques adaptés au cas bidimensionnel, (b) une estimation de l'aire inoccupée dans le conteneur par la résolution de plusieurs problèmes *subset sum*, (c) le calcul de bornes inférieures basées sur les *data-dependent dual-feasible functions* [82], (d) l'élimination de certaines symétries de blocs présentées dans Scheithauer [83]. Cette méthode présente l'avantage de détecter efficacement les instances insatisfiables.
Beldiceanu et Carlsson [84] utilisent une contrainte globale de non-chevauchement qui est le cas bidimensionnel de la contrainte `diffn` [85] et assure donc le respect des contraintes de non-chevauchement entre articles. La contrainte utilise un algorithme à balayage basée sur la notion de régions interdites qui représente l'ensemble des origines du rectangle i telles qu'il chevauche obligatoirement le rectangle j. À chaque appel, l'algorithme de filtrage vérifie qu'il existe une position n'appartenant à aucune région interdite pour chaque borne d'un article. Dans le cas contraire, on met à jour la borne concernée grâce à l'algorithme de balayage. Beldiceanu *et al.* [86] ont étendu l'utilisation des algorithmes à balayage pour le traitement de problèmes placement complexes (multidimensionnel, polymorphisme ...) en combinaison avec d'autres algorithmes de filtrage (cumulatif, estimation de l'espace inoccupée ...).

3.6 Conclusion

Ainsi, les problèmes d'ordonnancement et de placement sont intensivement étudiés par les chercheurs en programmation par contraintes. La diversité des approches proposées permet de traiter de nombreux problèmes mais demande une expertise certaine pour déterminer les modèles et techniques appropriées à la résolution d'un problème particulier. Nous verrons dans la seconde partie de cette thèse qu'il en va de même pour les stratégies de recherche.