Techniques de décisions hiérarchiques pour l'ordonnancement de tâches

Adriana Pacheco, Cédric Pralet, Stéphanie Roussel

Techniques de décisions hiérarchiques pour l'ordonnancement de tâches

Adriana Pacheco[*], **Cédric Pralet**, **Stéphanie Roussel**
ONERA / DTIS, Université de Toulouse, F-31055 Toulouse – France

prénom.nom@onera.fr

Résumé

Dans cet article, nous nous intéressons à des problèmes d'ordonnancement dans lesquels les tâches candidates sont hiérarchisées. Nous commençons par définir formellement un cadre de modélisation des problèmes d'ordonnancement hiérarchique. Nous proposons une première traduction de ce cadre vers la programmation par contraintes, ainsi qu'une méthode de décision basée sur des mécanismes d'abstractions et de décompositions des tâches du problème. Nous présentons différentes heuristiques d'abstraction et de décomposition associées à cette deuxième méthode. Ces deux méthodes ont été implémentées et nous montrons les résultats préliminaires obtenus sur des benchmarks représentatifs d'une application multi-robots.

Abstract

In this article, we consider scheduling problems in which candidate tasks are hierarchical. Fist of all, we formally define a framework for modelling hierarchical scheduling problems. We propose a first interpretation of this framework towards constraint programming, as well as a decision method based on abstraction and decomposition mechanisms of the problem tasks. We present several heuristics associated with the second method for the abstraction and the decomposition step. These two methods have been implemented and we present the obtained results on representative benchmarks of a multi-robot application.

1 Introduction

De nombreuses applications qui présentent un problème de décision ou d'optimisation combinatoire peuvent être formalisées à l'aide de modèles « tâches-ressources ». Ces derniers reposent sur un ensemble de tâches candidates et un ensemble de ressources disponibles pour les réaliser, et optimisent certains critères liés au temps ou aux ressources. C'est par exemple le cas des problèmes d'exploration multi-robots dans lesquels les tâches sont les différentes observations à réaliser et les déplacements associés à ces observations et les ressources sont les robots, les fenêtres ou canaux de communication qu'ils utilisent, le terrain sur lequel ils évoluent, l'énergie dont ils disposent, *etc.* (voir [1] pour une description d'un cas d'application). C'est également le cas des problèmes de planification des activités de satellites dans lesquels les tâches sont les observations et les vidages que le satellite doit réaliser et les ressources sont les instruments du satellite et les stations sol (vue d'ensemble de l'état de l'art présentée dans [4]). On peut finalement citer des problèmes d'allocation de fonctions sur des architectures avioniques où les tâches sont les fonctions avioniques et les ressources sont les calculateurs et le réseau de communication physique entre ces derniers ([7]).

En pratique, dans les problèmes listés ci-dessus, il est souvent utile de raisonner à différents niveaux d'abstraction. Dans le cas des applications de type exploration multi-robots, on décide par exemple à haut niveau de l'allocation des zones à explorer à chaque robot, puis à plus bas niveau de l'ordre dans lequel les observations associées à chaque zone sont réalisées et encore à plus bas niveau de la trajectoire fine de chacun des robots.

Pour prendre en compte ces niveaux d'abstraction, une première approche souvent utilisée en pratique consiste à décomposer explicitement le problème à résoudre en plusieurs sous-problèmes et à définir pour chacun de ces sous-problèmes une technique de résolution dédiée. Une seconde approche consiste à utiliser des solutions plus génériques, comme cela est fait

[*]Papier doctorant : Adriana Pacheco est auteur principal.

en planification avec le cadre des HTNs (Hierarchical Task Networks [6, 8, 2]), dans lequel le problème générique considéré est de décomposer des tâches de haut niveau en tâches dites atomiques, en utilisant un catalogue de méthodes de décomposition de tâches fourni en entrée.

Le cadre originel des HTNs n'est pas adapté pour modéliser les aspects « tâches-ressources ». En effet, la consommation des ressources par les tâches sont dans ce cas modélisées par des paramètres de l'état courant du système pouvant rendre complexe des raisonnements sur l'utilisation de ces ressources à différents niveaux de la hiérarchie de décision. Plusieurs travaux récents ([5, 10]) étendent le cadre HTN pour prendre en compte la notion de ressource. A l'inverse, peu de travaux cherchent à étendre le modèle classique « tâches-ressources » en y intégrant des aspects hiérarchiques.

Dans ce papier, nous suivons cette dernière approche et proposons un premier cadre de modélisation pour le problème d'ordonnancement hiérarchique défini par un ensemble de tâches pouvant être *atomiques* (i.e. pouvant être réalisées directement) ou *composite* (i.e. se décomposant en un sous-ensemble de tâches) et un ensemble de ressources disponibles pour les réaliser. Le problème d'ordonnancement hiérarchique s'intéresse au calcul de dates d'exécution optimales, en tenant compte des contraintes temporelles et des contraintes sur la disponibilité des ressources requises. Dans ce papier, nous nous intéressons plus particulièrement aux problèmes d'ordonnancement disjonctif, c'est-à-dire dans lesquels chaque ressource ne peut exécuter qu'une tâche à la fois.

Dans la section 2, nous définissons formellement le type de problèmes considérés ainsi qu'une première stratégie de résolution utilisant la programmation par contraintes. Pour réduire la combinatoire, nous introduisons des mécanismes d'abstraction des tâches composites du problème dans la section 3. Un algorithme de décomposition itérative des tâches du problème est ensuite proposé dans la section **??**. Les méthodes définies ont été implémentées et nous montrons les résultats obtenus sur des benchmarks représentatifs d'une application multi-robots dans la section 5. Nous concluons en section 6 sur les perspectives de ce travail et de la thèse associée.

2 Problèmes d'ordonnancement hiérarchique

Dans cette partie, nous définissons formellement la classe des problèmes d'ordonnancement hiérarchique auxquels nous nous intéressons. Des classes plus générales pourraient être considérées, mais nous traitons une classe relativement simple pour réaliser nos premières études.

Modèle général Un problème d'ordonnancement hiérarchique est un tuple $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ dans lequel :

- $\mathcal{R}$ est un ensemble de *ressources disjonctives* (pouvant réaliser au plus une activité à la fois) ;
- $\mathcal{A}$ est un ensemble d'*activités* à réaliser, aussi appelées des *tâches atomiques* ; chaque activité $a \in \mathcal{A}$ est définie par une durée du_a et par l'ensemble des ressources $\mathcal{R}_a \subseteq \mathcal{R}$ qu'elle consomme pendant toute cette durée ;
- $\mathcal{C}$ est un ensemble de *tâches composites* ; chaque tâche composite $c \in \mathcal{C}$ est définie par :
  - un ensemble de sous-tâches $SubTsk_c \subset \mathcal{C} \cup \mathcal{A}$;
  - un ensemble de ressources $\mathcal{R}_c \subseteq \mathcal{R}$ consommées tant que la tâche composite est active (c'est-à-dire de la date de début de la première sous-tâche associée à c jusqu'à la date de fin de la dernière sous-tâche associée à c) ;
  - un ensemble de contraintes de précédence acycliques $\mathcal{P}_c$ entre les sous-tâches de c ($\mathcal{P}_c \subset SubTsk_c \times SubTsk_c$).

Dans ce qui suit, pour toute ressource $r \in \mathcal{R}$, on note $\mathcal{T}_r$ l'ensemble des tâches τ (atomiques ou composites) qui utilisent la ressource r, c'est-à-dire telles que $r \in \mathcal{R}_\tau$.

Hypothèses additionnelles Nous supposons que les problèmes d'ordonnancement hiérarchiques auxquels on s'intéresse sont *bien formés*. Plus précisément, on suppose que :

- le graphe de décomposition des tâches est acyclique ; formellement, ce graphe est le graphe orienté dont les nœuds sont les tâches de $\mathcal{A} \cup \mathcal{C}$, et qui contient pour chaque tâche composite c et chaque sous-tâche $\tau \in SubTsk_c$ un arc $c \to \tau$;
- chaque tâche atomique ou composite apparaît comme sous-tâche d'au plus une tâche ; cela implique que le graphe de décomposition des tâches définit une forêt de tâches ;
- une ressource déclarée comme étant consommée par une tâche composite c n'est pas déclarée comme consommée par une tâche τ descendante de c. Formellement, pour toute tâche composite c et toute tâche τ telles qu'il existe un chemin c vers τ dans le graphe de décomposition des tâches, on a $\mathcal{R}_c \cap \mathcal{R}_\tau = \emptyset$.

Commentaires sur le modèle Le modèle obtenu fait intervenir des décompositions hiérarchiques. Il permet de représenter facilement des problèmes de types *Job Shop Scheduling* ou *Open Shop Scheduling* [9], avec dans ce cas un seul niveau de décomposition (une tâche

composite par job). Dans le cas du *Job Shop*, la modélisation fait apparaître des contraintes de précédence définissant les séquences d'activités associées aux différents jobs. Dans le cas *Open Shop*, où l'ordre des tâches d'un job est laissé libre, la modélisation fait également apparaître une ressource factice supplémentaire pour interdire que des tâches d'un même job soient réalisées en parallèle.

De manière orthogonale, dans la mesure où chaque tâche peut consommer plusieurs ressources, le modèle couvre également les RCPSP (Resource Constrained Project Scheduling Problems [3]). Pour le moment nous ne considérons que des ressources de capacité unitaire, et l'extension aux ressources cumulatives est laissée pour des travaux futurs. Enfin, il est important de noter que les consommations de ressources peuvent être associées directement aux tâches composites, ce qui permet de modéliser des scénarios impliquant des ressources monopolisées pendant toute la durée d'un ensemble de sous-tâches, par exemple un scénario impliquant un opérateur devant réaliser tout seul un ensemble de travaux (tout seul pour éviter des "changements de contexte") mais devant attendre que des outillages partagés entre plusieurs opérateurs soient disponibles.

Exemple Nous nous intéressons à un problème d'ordonnancement hiérarchique basé sur une application d'exploration multi-robots, dans lequel des robots doivent se déplacer sur un terrain et réaliser un certain nombre d'observations de zones. Plus précisément, on attribue aux robots des zones à observer. Chaque observation de zone se décompose en un ensemble d'observations atomiques à réaliser en séquence, entre lesquelles le robot doit se déplacer. Ces déplacements peuvent eux-mêmes être décomposés en points de passage à rallier séquentiellement. On suppose que les robots ne peuvent pas observer plus d'une zone à la fois. Ils doivent de plus transmettre en temps réel chaque observation atomique au centre de mission, et pour cela ils utilisent une fréquence d'émission spécifique liée à la nature de l'observation réalisée. Finalement, les points de passage ne peuvent pas être occupés par plus d'un robot à chaque pas de temps.

Un exemple jouet associé à cette application est illustré sur la figure 1 :

- deux zones Z_0 et Z_1 doivent respectivement être explorées par deux robots r_0 et r_1 ;
- pour observer la zone Z_0, le robot r_0 doit d'abord réaliser l'observation O_0 et la retransmettre sur la fréquence f_0, puis réaliser O_1 sur f_1 ;
- le déplacement M_{01} se décompose en 3 étapes : M^0_{01} sur le point p_0, M^1_{01} sur le point p_1 et M^2_{01} sur le point p_2 ;
- un découpage similaire est réalisé pour la zone Z_1.

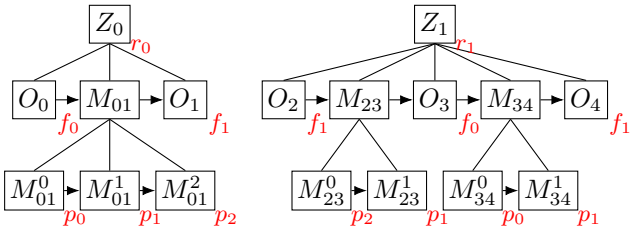

FIGURE 1 – Problème d'exploration multi-robots

Avec le cadre défini précédemment, cet exemple jouet se modélise comme suit :

- $\mathcal{R} = \{r_0, r_1, f_0, f_1, p_0, p_1, p_2\}$;
- $\mathcal{A} = \{O_0(2), O_1(3), O_2(3), O_3(1), O_4(2), M^0_{01}(1), M^1_{01}(2), M^2_{01}(1), M^0_{23}(2), M^1_{23}(1), M^0_{34}(2), M^1_{34}(1)\}$ (la durée de chaque activité est ici précisée entre parenthèses) ;
- $\mathcal{C} = \{Z_0, Z_1, M_{01}, M_{23}, M_{34}\}$;
- $SubTsk_{Z_0} = \{O_0, M_{01}, O_1\}$;
- $\mathcal{P}_{Z_0} = \{(O_0, M_{01}), (M_{01}, O_1)\}$;
- $SubTsk_{M_{01}} = \{M^0_{01}, M^1_{01}, M^2_{01}\}$;
- $\mathcal{P}_{M_{01}} = \{(M^0_{01}, M^1_{01}), (M^1_{01}, M^2_{01})\}$;
- $SubTsk_{Z_1} = \{O_2, M_{23}, O_3, M_{34}, O_4\}$;
- $\mathcal{P}_{Z_1} = \{(O_2, M_{23}), (M_{23}, O_3), (O_3, M_{34}), (M_{34}, O_4)\}$;
- $SubTsk_{M_{23}} = \{M^0_{23}, M^1_{23}\}$;
- $\mathcal{P}_{M_{23}} = \{(M^0_{23}, M^1_{23})\}$;
- $SubTsk_{M_{34}} = \{M^0_{34}, M^1_{34}\}$;
- $\mathcal{P}_{M_{34}} = \{(M^0_{34}, M^1_{34})\}$;
- $\mathcal{T}_{r_0} = \{Z_0\}$; $\mathcal{T}_{r_1} = \{Z_1\}$;
- $\mathcal{T}_{f_0} = \{O_0, O_3\}$; $\mathcal{T}_{f_1} = \{O_1, O_2, O_4\}$;
- $\mathcal{T}_{p_0} = \{M^0_{01}, M^0_{34}\}$;
- $\mathcal{T}_{p_1} = \{M^1_{01}, M^1_{23}, M^1_{34}\}$;
- $\mathcal{T}_{p_2} = \{M^2_{01}, M^0_{23}\}$.

L'exemple présenté fait intervenir des ressources utilisées pendant l'exécution des tâches d'intérêt (les observations utilisant des robots et des fréquences de communication) et un réseau partagé utilisé pour la mise au point de certaines ressources (les ressources robots qui doivent se déplacer dans l'environnement via des points de passage). La problématique serait la même pour des applications d'ordonnancement de fonctions sur une architecture embarquée, rencontrées par exemple en aéronautique, avec dans ce cas des ressources calculateurs utilisées pour exécuter les tâches d'intérêt (des calculs liées aux lois de commande, à l'estimation de la position d'un engin...) et un réseau embarqué constitué de liens réseaux et de nœuds réseaux à partager lors de la transmission de données entre certaines fonctions calculées.

Solution Une solution pour un problème d'ordonnancement hiérarchique associe une date de début s_τ et une date de fin e_τ à chaque tâche τ de $\mathcal{A} \cup \mathcal{C}$. Plusieurs contraintes doivent être satisfaites : pour chaque activité (ou tâche atomique), la date de fin de l'activité est donnée par la somme de sa date de début et de sa durée (équation 1) ; les dates de début et de fin d'une tâche composite c doivent couvrir exactement les sous-tâches de c (équations 2-3) ; deux tâches consommant la même ressource ne peuvent pas se chevaucher temporellement (équation 4) ; une précédence entre deux tâches implique que la date de fin de la première tâche doit être inférieure ou égale à la date de début de la seconde (équation 5).

$$\forall a \in \mathcal{A},\ e_a = s_a + du_a \quad (1)$$
$$\forall c \in \mathcal{C},\ s_c = \min\{s_\tau \mid \tau \in SubTsk_c\} \quad (2)$$
$$\forall c \in \mathcal{C},\ e_c = \max\{e_\tau \mid \tau \in SubTsk_c\} \quad (3)$$
$$\forall r \in \mathcal{R}, \forall \tau \neq \tau' \in \mathcal{T}_r^2,\ (e_\tau \leq s_{\tau'}) \vee (e_{\tau'} \leq s_\tau) \quad (4)$$
$$\forall c \in \mathcal{C},\ \forall (\tau, \tau') \in \mathcal{P}_c,\ s_{\tau'} \geq e_\tau \quad (5)$$

Une solution est dite optimale si elle minimise le *makespan*, défini comme la date de fin de la dernière tâche du plan ($\max\{e_\tau \mid \tau \in \mathcal{A} \cup \mathcal{C}\}$).

La figure 2 illustre la consommation des ressources associée à une solution de l'exemple jouet respectant les différentes contraintes du problème. Notons que sur cette figure, les tâches M_{01}, M_{23} et M_{34} n'apparaissent pas car elles ne consomment pas directement de ressources. Leurs dates de début et de fin sont : $s_{M_{01}} = 2$, $e_{M_{01}} = 6$, $s_{M_{23}} = 3$, $e_{M_{23}} = 6$, $s_{M_{34}} = 7$, $e_{M_{34}} = 10$.

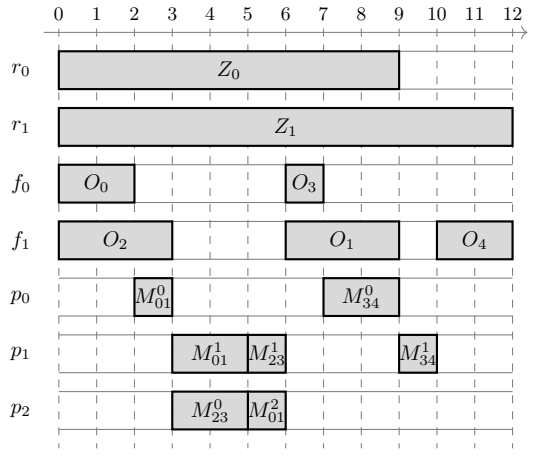

FIGURE 2 – Consommation des ressources pour une solution du problème jouet

Résolution en programmation par contraintes L'encodage en programmation par contraintes du problème d'ordonnancement hiérarchique défini à la section précédente est relativement direct. Il est par exemple possible de réutiliser les primitives d'ordonnancement disponibles dans l'outil IBM ILOG CpOptimizer [1], dont certaines servent justement à définir des hiérarchies de tâches. La modélisation obtenue est présentée ci-après. Elle fait tout d'abord intervenir, pour chaque tâche τ du problème, un intervalle temporel itv_τ (appelée une *variable intervalle* dans CpOptimizer). Chaque intervalle itv est défini par une variable $startOf(itv)$ représentant la date de début de l'intervalle, une variable $endOf(itv)$ représentant la date de fin de l'intervalle, et une longueur $lengthOf(itv)$ donnant la distance entre le début et la fin de l'intervalle. Pour les intervalles associés aux activités, cette distance est connue initialement et égale à la durée de l'activité (voir équation 6). Pour les intervalles associés aux tâches composites, cette distance n'est pas connue initialement (voir équation 7), et il est seulement possible de spécifier que les intervalles en question doivent se finir avant un horizon temporel maximum T considéré.

Sur cette base, on retrouve ensuite dans le modèle obtenu les différentes contraintes listées précédemment dans les équations 1 à 5. La formalisation utilise notamment la contrainte *span* spécifiant qu'un intervalle doit couvrir exactement un ensemble d'intervalles, la contrainte *noOverlap* imposant un non chevauchement temporel entre des intervalles, et la contrainte *endBeforeStart* imposant que la date de fin d'un intervalle soit inférieure ou égale à la date de début d'un autre intervalle. Le problème ainsi modélisé peut ensuite être résolu à l'aide de l'outil CpOptimizer.

Variables :

$$\forall a \in \mathcal{A},\ \textbf{dvar interval}\ itv_a \,\text{size}\, du_a \,\text{in}\, [0..\text{T}] \quad (6)$$
$$\forall c \in \mathcal{C},\ \textbf{dvar interval}\ itv_c \,\text{in}\, [0..\text{T}] \quad (7)$$

Constraints :

$$\forall c \in \mathcal{C},\ span(itv_c, \{itv_\tau \mid \tau \in SubTsk_c\}) \quad (8)$$
$$\forall r \in \mathcal{R},\ noOverlap(\{itv_\tau \mid \tau \in \mathcal{T}_r\}) \quad (9)$$
$$\forall c \in \mathcal{C},\ \forall (\tau, \tau') \in \mathcal{P}_c, endBeforeStart(itv_\tau, itv_{\tau'}) \quad (10)$$

3 Abstractions d'une tâche composite

La méthode de programmation par contraintes présentée précédemment s'appuie sur une version complètement dépliée du réseau de tâches hiérarchiques. Ce dépliage complet peut cependant être coûteux du point de vue de la résolution lorsque le nombre de tâches atomiques et composites augmente. Pour faciliter le passage à l'échelle et guider la résolution à plus haut niveau, nous définissons une méthode qui évite le dépliage systématique de toutes les tâches au début de

1. https://www-01.ibm.com/software/commerce/optimization/cplex-cp-optimizer/

la recherche. La méthode introduite raisonne sur des *abstractions des tâches composites* dans un premier temps, et elle *raffine* ensuite pas à pas ces abstractions lorsque cela s'avère nécessaire, en revenant progressivement à une modélisation non abstraite pour certaines tâches composites.

Dans ce contexte, abstraire une tâche composite signifie ne pas représenter finement toutes les sous-tâches qui la composent et raisonner à plus gros grain en estimant l'impact global de ces sous-tâches sur le problème d'ordonnancement à résoudre. Plus précisément, nous cherchons à abstraire chaque tâche composite c par une tâche atomique notée $Abs(c)$ définie simplement par une durée $du_{Abs(c)}$ et par un ensemble ressources $\mathcal{R}_{Abs(c)}$ consommées pendant toute la durée de c. Nous détaillons ci-après les techniques utilisées pour définir les quantités $du_{Abs(c)}$ et $\mathcal{R}_{Abs(c)}$. Les abstractions sont calculées en partant des tâches atomiques (avec pour toute tâche atomique $a \in \mathcal{A}$ la convention $Abs(a) = a$) et se propagent progressivement jusqu'aux tâches de plus haut niveau dans la hiérarchie des tâches.

3.1 Durée de l'abstraction d'une tâche composite

Pour définir la durée $du_{Abs(c)}$ associée à l'abstraction d'une tâche composite $c \in \mathcal{C}$, il est tout d'abord possible de considérer le problème de l'ordonnancement des abstractions des sous-tâches de c indépendamment des autres tâches composites du problème. Ce problème Pb contient une activité $Abs(\tau)$ pour chaque sous-tâche $\tau \in SubTsk_c$, et une contrainte de précédence $Abs(\tau) \rightarrow Abs(\tau')$ pour chaque contrainte de précédence $\tau \rightarrow \tau' \in \mathcal{P}_c$. On suppose ensuite que l'on dispose d'une procédure capable de produire rapidement une solution S au problème Pb. Cette procédure peut par exemple correspondre à l'utilisation d'une règle heuristique qui insère les tâches les unes après les autres dans le plan en suivant un ordre d'insertion fonction des caractéristiques des tâches du problème. Lorsque le problème Pb reste simple, elle peut également correspondre à une résolution à l'aide d'un outil de résolution complet capable de produire un ordonnancement minimisant le makespan. Notons que dans le cas où les contraintes de précédence associées à c sont telles que toutes les sous-tâches de c doivent être réalisées en séquence, obtenir une solution S minimisant le makespan est immédiat (dans ce cas, le makespan optimal vaut $\sum_{\tau \in SubTsk_c} du_{Abs(\tau)}$).

La solution S trouvée en ordonnançant les abstractions des sous-tâches de c peut ensuite être utilisée pour associer à l'abstraction de c une durée $du_{Abs(c)}$ égale au makespan $mk(S)$ de la solution S. Intuitivement, cette abstraction est pessimiste dans le sens où elle considère une durée qui est de toutes façons suffisante pour réaliser l'intégralité de la tâche c.

La figure 3 montre une tâche composite de départ et un plan minimisant le makespan qu'il est possible d'obtenir directement vu les contraintes de précédence. La solution donnée à la figure 3(b) donnerait une durée égale à 4 unités de temps avec l'abstraction choisie.

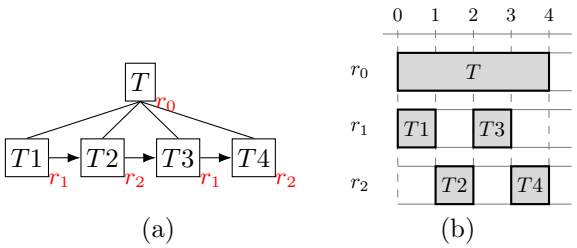

FIGURE 3 – Ordonnancement sur une tâche composite individuelle : (a) tâche composite T de départ faisant intervenir 4 sous-tâches de durée 1, pour un problème impliquant trois ressources r_0, r_1, r_2 ; (b) ordonnancement solution pour le problème restreint à la tâche T

3.2 Consommations de ressource pour l'abstraction d'une tâche composite

Pour abstraire les consommations de ressources provenant des sous-tâches de c, nous considérons deux versions :

- une version notée $C0$, donnée à l'équation 11, dans laquelle on inclut dans les consommations de $Abs(c)$ uniquement les ressources de $\mathcal{R}_c$ qui sont consommées directement par c ;
- une version notée $C1$, donnée à l'équation 12, dans laquelle on ajoute dans les consommations de $Abs(c)$ l'ensemble des ressources consommées par les abstractions des sous-tâches de c (nous rappelons que les abstractions sont construites en suivant une approche *bottom-up* dans la hiérarchie des tâches).

Intuitivement, la version $C0$ est une version plutôt optimiste dans laquelle on considère que de toutes façons les consommations des ressources par les sous-tâches ne seront pas limitantes pour l'ordonnancement (sur l'exemple du dernier niveau de hiérarchie pour le problème de déploiement de robots, cette approche revient à considérer en première approximation que la ressource réseau ne freinera pas les déplacements des robots). La version $C1$ est quant à elle une version plus pessimite (ou *robuste*) dans le sens où elle réserve pendant toute la durée de l'abstraction toutes les ressources qui pourraient être consommées par des sous-tâches.

$$C0 \ : \ \mathcal{R}_{Abs(c)} = \mathcal{R}_c \tag{11}$$

$$C1 \ : \ \mathcal{R}_{Abs(c)} = \mathcal{R}_c \cup (\cup_{\tau \in SubTsk_c} \mathcal{R}_{Abs(\tau)}) \tag{12}$$

Nous obtenons ainsi deux méthodes d'abstraction à étudier, notées respectivement Abs^{C0}, Abs^{C1}. Ces abstractions sont illustrées à la figure 4. L'approche Abs^{C1} conduit à des ordonnancements de tâches qui d'une certaine manière cloisonnent les consommations de ressources par les tâches. Cette stratégie permet de construire des ordonnancements qui ne contiennent aucune *interférence* entre les sous-tâches de tâches composites différentes. Cela a également pour effet de garantir que l'ordonnancement construit sur la base des abstractions peut à tout instant être étendu à un ordonnancement complet. Dans ce sens, l'abstraction Abs^{C1} est *robuste*. A l'opposé, l'abstraction Abs^{C0} est optimiste.

Notons enfin que les abstractions obtenues raisonnent implicitement comme si les sous-tâches de tâches différentes ne pouvaient pas être *entrelacées* les unes avec les autres sur les ressources, c'est-à-dire en ne considérant pas de schéma d'exécution dans lequel une ressource est utilisée d'abord pour les sous-taches d'une tâche composite c_1, puis par des sous-tâches d'une autre tâche composite c_2, puis à nouveau par des sous-tâches de c_1. Les optimisations utilisant de l'entrelacement de tâches sur les ressources sont laissées à des phases ultérieures de la recherche, au cours desquelles les tâches abstraites sont raffinées si cela s'avère utile.

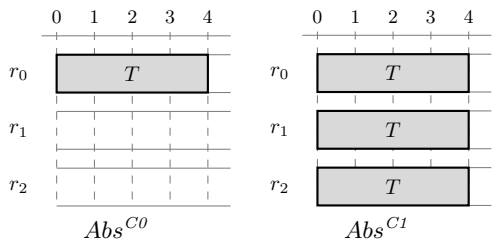

FIGURE 4 – Différentes formes d'abstraction obtenues à partir de la solution de la figure 3

3.3 Remarques

La démarche globale utilisée fait le choix d'essayer de mettre à profit la hiérarchie de décomposition des tâches pour guider la recherche. La hiérarchie de décomposition est ainsi exploitée au niveau algorithmique, et pas seulement pour des raisons de modélisation. D'autres méthodes seraient envisageables, comme chercher à grouper automatiquement des tâches du problème qui sont fortement contraintes les unes par rapport aux autres, réaliser des abstractions par catégories de ressources... Par ailleurs, la méthode utilisée explore l'utilisation d'abstractions simples en définissant une durée et un ensemble de consommations pour chaque tâche composite. D'autres méthodes utilisant une vue plus détaillée de l'ordonnancement solution trouvé pour chaque tâche composite pourront être explorées dans des travaux futurs. Enfin, notons que les abstractions sont calculées sur la base d'un ordonnancement qui correspond à une solution effective pour chaque tâche composite, contrairement à une approche qui raisonnerait uniquement par propagation de contraintes pour obtenir des bornes inférieures sur la durée de ces tâches.

4 Stratégies de décomposition itérative

Nous définissons maintenant la stratégie globale de raffinement utilisée pour passer progressivement d'un plan construit pour un problème P_0 qui contient uniquement les tâches atomiques et les abstractions des tâches de haut niveau, à un plan construit pour un problème P_n qui correspond au problème d'origine $(\mathcal{R}, \mathcal{A}, \mathcal{C})$. Pour passer du problème P_i au problème raffiné P_{i+1}, le principe adopté consiste à sélectionner à chaque étape de l'algorithme un ensemble non vide de tâches composites qui sont présentes dans P_i sous leur forme abstraite, et à raffiner ces tâches de telle sorte que le nouveau problème P_{i+1} obtenu contienne au moins une tâche abstraite en moins. Différents réglages doivent être faits pour définir exactement comment passer du problème P_i au problème P_{i+1}.

Nombre de tâches à raffiner Le premier réglage à réaliser concerne le nombre de tâches abstraites qui sont raffinées à chaque étape, avec comme réglages possibles la sélection d'une seule tâche abstraite à chaque étape ou la sélection de K tâches abstraites à raffiner simultanément.

Heuristique de sélection des tâches à raffiner Le second réglage à réaliser concerne le choix d'une heuristique permettant de privilégier le raffinement des tâches abstraites les plus intéressantes. Dans les expérimentations, nous étudions trois heuristiques de sélection, notées respectivement $H1$, $H2$ et $H3$. Ces abstractions sont illustrées :

— la première heuristique privilégie les tâches abstraites dont la date de début est minimale ; cette heuristique est notamment bien adaptée pour les approches dites en ligne dans lesquelles l'exécution du plan est réalisée en parallèle de la planification, et donc pour lesquelles les premières actions du plan doivent être engagées à un certain instant potentiellement proche ;

— la deuxième heuristique privilégie les tâches abstraites situées au plus haut niveau de la hiérarchie, c'est-à-dire les tâches abstraites pour

lesquelles le nombre de tâches composites ascendantes est minimal ; cette approche reprend quelque part des principes d'une approche classique de gestion des problèmes d'ordonnancement hiérarchique, consistant à traiter d'abord complètement le problème de décision de plus haut niveau avant d'aborder les spécifications plus fines du problème ;

— la troisième heuristique privilégie les tâches abstraites qui apparaissent sur un chemin critique dans le plan solution S trouvé pour le problème P_i, c'est-à-dire les tâches abstraites qui ont une flexibilité temporelle nulle dans S ; l'objectif est ici de concentrer d'abord la recherche sur les points durs du problème.

Pour toutes les heuristiques, s'il existe des tâches abstraites ex æquo au sens des critères utilisés par l'heuristique, alors ces tâches sont départagées aléatoirement.

Informations transmises entre les itérations Pour que la résolution du problème pas à pas présente un intérêt plutôt que le raisonnement direct sur le problème complet, il est utile de transmettre des informations entre les résolutions successives. Autrement dit, il est utile de définir des méthodes pour que les premières résolutions guident les résolutions ultérieures, ces dernières pouvant en effet se retrouver plus facilement coincées dans certaines zones d'un espace de recherche potentiellement beaucoup plus grand suite au raffinement. Deux types de transmissions d'information sont envisagées :

— soit la transmission d'une contrainte imposant que le makespan obtenu pour le problème P_{i+1} soit inférieur ou égal au makespan de la solution trouvée pour le problème P_i ; cette contrainte est effectivement utilisée uniquement lorsque les abstractions choisies sont pessimistes, car dans le cas contraire il n'est pas garanti que le makespan optimal de P_{i+1} puisse être inférieur ou égal au makespan de P_i ;

— soit la transmission de contraintes de précédence de type "start-to-start" entre tâches, l'intuition étant que les résolutions gros grain ont potentiellement permis de synthétiser de bons ordres d'utilisation des ressources par les tâches ; de manière plus précise, partant du plan solution trouvé pour le problème P_i, on extrait pour chaque ressource $r \in \mathcal{R}$ toutes les tâches de P_i qui utilise r, on classe ces tâches par date de début croissante, et pour toute paire de tâches τ, τ' qui se succèdent dans cet ordre, on ajoute à P_{i+1} la contrainte $s_\tau \leq s_{\tau'}$ imposant que τ commence avant τ' si cette contrainte n'était pas déjà présente dans P_i.

Algorithme de décomposition itérative L'algorithme 1 décrit la méthode globale utilisée. Dans cette dernière, on retrouve une phase initiale de construction du problème où toutes les tâches composites sont abstraites (ligne 1), une résolution de ce problème avec un temps de calcul maximum *MaxTimeIter* donné en entrée de l'algorithme (ligne 2), puis des décompositions itératives utilisées tant qu'il reste des tâches abstraites (lignes 3 à 7). Lors de chaque étape de décomposition, l'algorithme sélectionne un ensemble de tâches abstraites à décomposer (ligne 4), extrait des contraintes à partir de la solution courante S (ligne 5), calcule un problème P actualisé (ligne 6) et résout ce nouveau problème toujours avec un temps de calcul maximum (ligne 7). La solution trouvée sur le dernier problème est enfin renvoyée (ligne 8). Ce pseudo-code pourrait être adapté pour répartir le temps de calcul global autrement qu'en allouant le même temps de calcul à chaque itération du raffinement. L'étude de réglages plus évolués du temps de calcul par itération est laissée pour des travaux ultérieurs.

Algorithm 1: $iterativeDecomp(\mathcal{R}, \mathcal{A}, \mathcal{C}, MaxTimeIter)$: algorithme de décomposition itérative pour un problème d'ordonnnancement hiérarchique

```
1 P ← fullAbstractProblem(R, A, C);
2 S ← solve(P, MaxTimeIter);
3 while getAbsTasks(P) ≠ ∅ do
4     ToDecompose ← selectAbsTasks(P);
5     Ctrs ← extractConstraints(S);
6     P ← update(P, ToDecompose, Ctrs);
7     S ← solve(P, MaxTimeIter);
8 return S;
```

5 Expérimentations

Dans cette partie, nous montrons les résultats préliminaires obtenus sur différents benchmarks avec les méthodes décrites précédemment.

5.1 Benchmarks

Problème d'exploration multi-robot De manière à tester les méthodes sur des problèmes de grande taille proches des applications visées, nous avons généré des benchmarks autour de l'exploration multi-robots décrite dans l'exemple de la section 2. Nous avons créé un générateur pour ce type de benchmarks qui prend en paramètre les éléments suivants :

— **nZones :** nombre de zones d'observations ;

— **nObsPerZone :** nombre d'observations nécessaires à l'observation de chaque zone ;

— **nRobots :** nombre de robots disponibles pour réaliser les observations de zone ;

— **nFréquences :** nombre de fréquences disponibles pour émettre pendant une observation ;
— **nPointsTransferts :** nombre de points de passage total ;
— **nPointsPerMove :** nombre de points de passage pour un seul transfert pour un robot ;

Il est également possible de paramétrer les bornes des durées des différentes tâches atomiques du problème.

Nous montrons les résultats sur deux instances représentatives. Nous nommons la première *5Z* (5 zones) : c'est une petite instance avec 5 zones à observer, 2 robots pour les réaliser, 3 observations par zone, 10 points de passage au total, 3 points de passage par déplacement et 5 fréquences disponibles. La deuxième instance est nommée *50Z* (50 zones) et correspond à une plus grande instance avec 50 zones, 4 robots pour les réaliser, 10 observations par zone, 15 points de passage au total, 3 points de passage par déplacement et 3 fréquences disponibles.

Problème d'Open shop et de Job shop Nous avons également testé les méthodes sur des benchmarks de la littérature, à savoir les problèmes de type *Open Shop* et *Job Shop* disponibles sur la page de E .Taillard [2]. Pour les problèmes d'*Open shop*, nous utilisons deux instances 10×10 et 20×20 [3]. Plus précisément, nous transformons l'instance 10×10 (resp. 20×20) en un problème hiérarchique à deux niveaux avec 10 (resp. 20) tâches composites, 10 (resp. 20) sous-tâches par tâche, et 10 (resp. 20) ressources disjonctives. Pour les problèmes de *Job Shop*, nous utilisons les instances de taille 20×20 et 100×20. Nous transformons l'instance 20×20 (resp. l'instance 100×100) en un problème hiérarchique à deux niveaux avec 20 (resp. 100) tâches composites, 20 sous-tâches par tâche, et 20 ressources disjonctives.

5.2 Résultats

Le tableau 1 présente les résultats pour les deux instances du problème multi-robot. Pour la résolution de ces problèmes, un temps de calcul maximum *MaxTime* est fixé à 1 minute pour l'instance *5Z* et à 5 minutes pour l'instance *50Z* du problème. Lors de la résolution avec abstraction, le nombre d'itérations *MaxTimeIter* décrit dans l'algorithme 1 correspond au *MaxTime* réparti équitablement entre toutes les itérations. Dans ces expérimentations, les informations transmises entre les itérations sont des précédences de type "start-to-start".

2. `http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html`
3. Plusieurs instances ont ces tailles. Nous considérons les premières à chaque fois.

Pour chaque résolution du problème abstrait en utilisant les heuristiques de décomposition décrites précédemment, il faut choisir le nombre de tâches abstraites à décomposer simultanément K. Nous exprimons ici ce nombre en pourcentage p du nombre total de tâches composites. Formellement, $K = \lceil p \cdot \mathbf{card}(\mathcal{C}) \rceil$. Le pourcentage p est indiqué entre parenthèses à côté de l'heuristique utilisée. Nous avons ici choisi de décomposer soit 5% des tâches à chaque itération, soit 20%. La colonne *TB* indique le temps en secondes pour trouver la meilleure solution et la colonne *TT* indique le temps total utilisé (en secondes). Par rapport à l'algorithme 1 présenté dans la section 4, on note *TT* la date à laquelle chaque solution S est calculée, et *TB* représente la date à laquelle on trouve le makespan de la solution finale. Autrement dit, le makespan de la solution n'est pas amélioré entre *TB* et *TT*. Étant donné que pour l'abstraction Abs^{C0}, le makespan obtenu à chaque itération ne peut pas être considéré comme une vraie borne supérieure, le *TB* obtenu n'est pas indiqué dans les tableaux suivants.

Inst.	Abs.	H(%)	BestMks	TB	TT
5Z	Sans Abstraire		**158***	0.03	17.4
	Abs^{C0}	H1(5)	158	-	31.2
		H1(20)	158	-	30.9
		H2(5)	158	-	34.3
		H2(20)	158	-	44.2
		H3(5)	158	-	15.7
		H3(20)	158	-	31.8
	Abs^{C1}	H1(5)	158	17.0	29.7
		H1(20)	158	12.4	37.6
		H2(5)	158	24.9	46.2
		H2(20)	158	31.9	43.2
		H3(5)	158	31.0	43.3
		H3(20)	158	24.3	45.3
50Z	Sans Abstraire		3785	28	*MaxTime*
	Abs^{C0}	H1(5)	3919	-	*MaxTime*
		H1(20)	3910	-	*MaxTime*
		H2(5)	3919	-	*MaxTime*
		H2(20)	3910	-	*MaxTime*
		H3(5)	3919	-	*MaxTime*
		H3(20)	3910	-	*MaxTime*
	Abs^{C1}	H1(5)	3919	297	*MaxTime*
		H1(20)	3910	299	*MaxTime*
		H2(5)	3919	300	*MaxTime*
		H2(20)	3910	298	*MaxTime*
		H3(5)	3919	298	*MaxTime*
		H3(20)	3910	299	*MaxTime*

TABLE 1 – Résultats pour le problème d'exploration multi-robots.

La ligne *Sans Abstraire* correspond à la résolution de la traduction du problème en Programmation par

Contraintes par l'outil CpOptimizer. Lorsque CpOptimizer a prouvé que la borne était l'optimum, nous l'indiquons avec le symbole $*$.

Les tableaux 2 et 3 présentent les résultats pour les deux instances du problème d'*Open Shop* et les deux instances du problème de *Job Shop* respectivement, avec un temps de calcul maximum de 10 secondes.

Inst.	Abs.	H(%)	BestMks	TB	TT
10×10	Sans Abstraire		**637***	0.67	1.16
	Abs^{C0}	H1(20)	637	-	1.88
		H2(20)	637	-	1.89
		H3(20)	637	-	1.86
	Abs^{C1}	H1(20)	637	8.82	9.35
		H2(20)	637	8.86	9.52
		H3(20)	637	8.68	9.68
20×20	Sans Abstraire		**1155***	0.66	1.57
	Abs^{C0}	H1(20)	1155	-	1.9
		H2(20)	1155	-	1.91
		H3(20)	1155	-	1.82
	Abs^{C1}	H1(20)	1155	9.30	9.75
		H2(20)	1155	9.40	9.81
		H3(20)	1155	9.35	9.79

TABLE 2 – Résultats du problème d'Open shop

Les résultats obtenus sont des résultats préliminaires dans le contexte de nos objectifs de recherche et nous serviront de guide pour les expériences futures. Ils révèlent que l'outil d'optimisation CpOptimizer arrive à trouver des bonnes solutions assez rapidement. En effet, même pour le plus gros problème testé, une première solution est trouvée en 28 secondes, mais cette solution n'est pas améliorée dans la suite de la résolution.

Sur des petits problèmes, notre méthode est capable de trouver le makespan optimal trouvé par CP Optimizer. En revanche, sur des plus grands problèmes, en rajoutant les contraintes de précédence de type "start-to-start" entre les résolutions successives, l'espace de recherche est coupé et on risque potentiellement, d'enlever la solution optimale de l'espace de recherche. Notons tout de même que les solutions sont proches du $BestMks$ trouvé pour le problème sans abstraire (3.5%).

Même si les deux abstractions ont des résultats similaires dans les tableaux présentés ici, il s'avère que leur comportement dans la phase de recherche est très différent. Par exemple, en utilisant l'heuristique de décomposition $H3(5\%)$ pour l'instance 50Z, la première borne obtenue pour l'abstraction Abs^{C0} est de 3713, et cette dernière est considérablement plus élevée pour l'abstraction Abs^{C1} avec une valeur de 8083.

L'écart entre CpOptimizer et la méthode basée sur l'abstraction et la décomposition vient certainement

Inst.	Abs.	H(%)	BestMks	TB	TT
20×20	Sans Abstraire		**1217***	0.02	1.87
	Abs^{C0}	H1(20)	1217	-	2.38
		H2(20)	1217	-	2.48
		H3(20)	1217	-	2.35
	Abs^{C1}	H1(20)	1217	9.47	9.85
		H2(20)	1217	9.54	9.91
		H3(20)	1217	9.48	9.98
100 $\times$20	Sans Abstraire		**5464***	0.99	3.09
	Abs^{C0}	H1(20)	5464	-	6.73
		H2(20)	5464	-	7.05
		H3(20)	5464	-	6.74
	Abs^{C1}	H1(20)	5464	9.56	10.0
		H2(20)	5464	9.57	10.0
		H3(20)	5464	9.47	10.0

TABLE 3 – Résultats du problème de Job shop

du fait que les abstractions utilisées sont très agressives dans leur manière de définir la consommation de ressources. Les travaux futurs consisteront à définir des abstractions plus fines.

6 Conclusion

Dans ce papier, nous avons défini un premier cadre de modélisation pour les problèmes d'ordonnancement hiérarchique. Nous avons proposé une traduction en programmation par contraintes et définissons une méthode basée sur des mécanismes d'abstraction et de décomposition des tâches. Les expérimentations montrent que cette dernière pourrait largement être améliorée en considérant des stratégies plus fines d'abstraction. Une piste serait de définir des abstractions sur des problèmes utilisant des ressources cumulatives (ressources avec une certaine capacité), et en conséquence pouvoir considérer le pourcentage de consommation d'une ressource par une sous-tâche. Cela permettra notamment de prendre en compte plus finement la consommation des ressources des sous-tâches. Il sera utile de réexaminer aussi, les informations qui sont transmises entre les itérations, pour que l'espace de recherche ne se retrouve pas largement découpé, comme c'est le cas avec les contraintes de précédence "start-to-start".

D'autres perspectives consistent à étendre le cadre de modélisation. On pourra par exemple prendre en compte les ressources flexibles, c'est-à-dire permettre à chaque tâche de consommer une ressource parmi un ensemble de ressources disponibles. On pourra également définir pour chaque tâche composite des méthodes de décomposition, comme dans le cadre HTN.

Références

[1] P. Bechon, M. Barbier, C. Grand, S. Lacroix, C. Lesire, and C. Pralet. Integrating planning and execution for a team of heterogeneous robots with time and communication constraints.

[2] P. Bechon, M. Barbier, G. Infantes, C. Lesire, and V. Vidal. Hipop : Hierarchical partial-order planning. In *In Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS'14)*, 2014.

[3] P. Brucker, A. Drexl, R. Möring, K. Neumann, and E. Pesch. Resource-constrained Project Scheduling : Notation, Classification, Models, and Methods. *European Journal of Operational Research*, 112(1) :3–41, 1999.

[4] J. Colomé, P. Colomer, J. Guàrdia, I. Ribas, J. Campreciós, T. Coiffard, .L Gesa, F. Martínez, and F. Rodler. Research on schedulers for astronomical observatories. In *Observatory Operations : Strategies, Processes, and Systems IV*, volume 8448, page 84481L. International Society for Optics and Photonics, 2012.

[5] F. Dvorak, R. Barták, A. Bit-Monnot, F. Ingrand, and M. Ghallab. Planning and acting with temporal and hierarchical decomposition models. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*, pages 115–121, 2014.

[6] K. Erol, J. Hendler, and D. S. Nau. HTN planning : Complexity and expressivity. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 1123–1128. AAAI Press, 1994.

[7] S. Girbal, D. G. Pérez, J. Le Rhun, M. Faugère, C. Pagetti, and G. Durrieu. A complete toolchain for an interference-free deployment of avionic applications on multi-core systems. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, pages 7A2–1–7A2–14, Sept 2015.

[8] D. Nau, T. C. Au, O. Ilghami, U. Kuter, D. Wu, F. Yaman, H. Munoz-Avila, and J. W. Murdock. Applications of shop and shop2. *IEEE Intelligent Systems*, 20(2) :34–41, March 2005.

[9] M. L. Pinedo. *Scheduling : Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

[10] C. Qi, D. Wang, H. Muñoz-Avila, P. Zhao, and H. Wang. Hierarchical task network planning with resources and temporal constraints. *Knowledge-Based Systems*, 133 :17–32, 2017.