Actes JFPC 2008

Modélisation de problèmes d'ordonnancement avec adversaire en QCSP$^+$

Marco Benedetti Arnaud Lallouet Jérémie Vautard

Université d'Orléans – LIFO
BP 6759, F-45067 Orléans
{prenom.nom}@univ-orleans.fr

Résumé

Dans ce papier, nous considérons les problèmes d'ordonnancement à ressources cumulatives en présence d'un adversaire. Il s'agit pour l'ordonnanceur de gérer les ressources disponibles de façon à rester en deçà d'une date limite, tandis qu'un adversaire peut modifier certains paramètres – comme par exemple la consommation en ressources de certaines tâches – dans une limite donnée. La question est donc de savoir s'il existe un ordonnancement *robuste*, c'est à dire dont le déroulement est garanti, quelque soit le comportement de l'adversaire.

Nous proposons une modélisation de ce problème en QCSP$^+$, un formalisme voisin des problèmes de satisfaction de contraintes quantifiées dans lequel la quantification peut être restreinte. Nous présentons la résolution de différentes instances au moyen du solveur QeCode.

1 Introduction

Les problèmes d'ordonnancement non-préemptifs classiques consistent à trouver des dates de départ pour un ensemble de tâches telles que toutes les contraintes de précédence et de ressources sont satisfaites. La théorie de l'ordonnancement a été largement étudiée en IA et en RO [14] [10], et beaucoup de cas intéressants ont été relevés. Par exemple, la prise en compte de l'incertitude liée à l'environnement est maintenant considérée comme une extention majeure de l'ordonnancement [12]. L'introducton d'un *adversaire* rend le problème encore plus critique, étant donné que les méthodes probabilistes sont inappropriées pour modéliser le caractère *malveillant* du comportement de l'adversaire.

Dans ce papier, nous modélisons des *problèmes d'ordonnancement avec adversaire* en utilisant le formalisme des QCSP$^+$[6], une extention des problèmes de constraintes quantifiées (QCSP) dans laquelle il est possible de restreindre les quantificateurs. Dans notre approche, l'adversaire n'est pas tout-puissant. Ainsi, les actions qu'il entreprend doivent satisfaire un certain nombre de contraintes qui définissent un *modèle de l'adversaire*. Ces contraintes sont censées représenter les limites d'un adversaire réel. Il est par exemple possible de concevoir un modèle dans lequel l'adversaire peut doubler la consommation de ressources d'au plus la moitié des tâches. Quelles tâches précises devrait-il perturber de manière à créer la pire attaque possible ? Le modèle de l'adversaire seul ne suffit pas à répondre à cette question. La pire attaque possible, si elle existe, invalide n'importe quel ordonnancement faisable au départ (par exemple en dépassant les capacités en ressources). Mais, les ordonnancement faisables au départ dépendent bien sûr du problème considéré. Il y a donc une interdépendance forte entre ce que l'ordonnanceur est capable de produire et la manière dont l'adversaire préfèrera attaquer.

Nous montrons que les QCSP$^+$ conviennent tant à modéliser qu'à résoudre de nombreuses variantes d'un tel problème avec adversaire. L'idée de base est de quantifier universellement les actions de l'adversaire (de manière à capturer toutes les attaques possibles), mais en posant une restriction (de manière à être consistant avec le modèle de l'adversaire). De façon analogue, les propositions d'ordonnancement sont soumise à une quantification existentielle restreinte, la restriction consistant à modéliser la faisabilité dudit ordonnancement.

2 Ordonnancement avec adversaire

2.1 Contexte

L'ordonnancement consiste à allouer une certaine quantité de ressources à un certain nombre de tâches en fonction du temps. Dans le domaine de la recherche opérationnelle, la littérature considère plusieurs types de problèmes, répertoriés dans [12].

Selon [3], l'ordonnancement dit *préemptif* si les tâches peuvent être interrompues puis reprises, *non-préemptif* sinon, et *élastique* si leur consommation de ressources peut varier de manière continue entre 0 et le maximum disponible jusqu'à ce qu'elles aient consommé suffisamment de ressources pour s'achever.

Du point de vue des ressources, l'ordonnancement est dit *disjonctif* si toutes les ressources ont une capacité de 1, et *cumulatif* sinon. Dans ce dernier cas, chaque ressource a une capacité limitée qui ne peut être dépassée à aucun moment.

Les problèmes décisionnels d'ordonnancement consistent à vérifier s'il existe un ordonnancement qui safisfait toutes les contraintes de ressources et de temps. Un problème d'optimisation d'ordonnancement impose en outre de minimiser une certaine fonction d'objectif — typiquement le temps total nécessaire. Dans ce papier, nous étendons la classe des problèmes décisionnels d'ordonnancement cumulatifs non-préemptifs.

De par son efficacité et sa généralité, la programmation par contraintes (PPC) est devenue un paradigme majeur pour traiter les problèmes d'ordonnancement [3]. Cette généralité est due à la possibilité de modéliser la plupart des problèmes pratiques simplement en cumulant des contraintes appropriées, tandis que l'efficacité résulte de l'intégration d'algorithmes venant de la recherche opérationnelle au coeur de l'implémentation des contraintes globales.

En PPC, un problème d'ordonnancement peut être modélisé de la manière suivante : Considérons un ensemble donné d'activités $A = \{a_1, \ldots, a_n\}$ à planifier. Une activité a_i est définie par sa date de début s_i, sa durée d_i et sa date de fin e_i, comme dans la figure 1. Étant donné que $s_i + d_i = e_i$, seulement deux variables sont nécessaires pour représenter une activité (bien qu'il soit parfois utile de les représenter toutes les trois pour des raisons de modélisation).

Deux types de contraintes sont présentes dans les problèmes d'ordonnancement : les contraintes temporelles et les contraintes de ressources.

D'une part, les contraintes temporelles définissent un ordre partiel sur les activités : par exemple, il peut arriver qu'une tâche a_i ne puisse débuter qu'après qu'une autre tâche a_j se soit terminée. Soit $(A, \prec)$ une

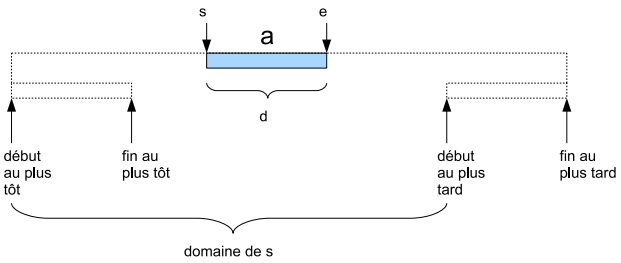

FIG. 1 – Représentation d'une activité

relation de précédence sur l'ensemble des activités, signifiant que pour tout $a_i, a_j \in A$, si $a_i \prec a_j$, alors a_i doit être terminée avant que a_j ne puisse démarrer. Cette contrainte de précédence se traduit naturellement par la contrainte $e_i \leq s_j$. Certaines contraintes, comme les dates au plus tôt ou au plus tard de début d'une tâche, peuvent être directement représentées en restreignant les domaines des variables correspondantes. Parfois, l'espace entre deux tâches se trouve contraint entre des valeurs minimum (min_{ij}) et maximum (max_{ij}).

D'autre part, le second type de contraintes rencontrées regroupe les contraintes de ressources. Soit $R = \{r_1, \ldots, r_m\}$ un ensemble de ressources. Chaque activité a_i demande une certaine quantité c_{ij} de ressources $r_j \in R$ pour s'exécuter. La capacité maximum de chaque ressources contraint donc le taux de concurrence qu'une solution peut proposer. Soit $rmax_i$ la capacité de la ressource r_i. Pour une seule ressource r_i, le problème peut aisément être modélisé grâce à la contrainte globale $Cumulative([s_1, \ldots, s_n], [d_1, \ldots, d_n], [c_{1i}, \ldots, c_{ni}], rmax_i)$ [1]. Cette contrainte garantit que, pour une ressource $r_j \in R$:

$$\forall t \geq 0, \left(\sum_{\{i \mid s_i \leq t \leq e_i\}} c_{ij} \right) \leq rmax_j$$

Les problèmes impliquant plusieurs ressources sont traités soit en posant une contrainte Cumulative par ressource, soit en utilisant une contrainte multi-ressources spécialisée [5]. Une ressource dont la capacité change au cours du temps peut être simulée en ajoutant des tâches "fantômes" à des dates fixées.

Dans beaucoup de problèmes, la durée des tâches est fixée et connue à l'avance. Dans ces cas là, le problème d'ordonnancement consiste à trouver une date de début pour toutes les tâches de manière à ce que le plan correspondant respecte toutes les contraintes temporelles et de ressources. De plus, une fonction de coût requiert souvent d'être minimisée, comme par exemple le « makespan »(date de fin de la dernière tâche du projet), ou la « tardiness »(ou le retard, c'est à dire

la différence entre le temps de terminaison effectif et attendu). Certains problèmes d'ordonnancement particuliers ont été étudiés, par exemple les problèmes de *job-shop*, qui sont utilisés dans les ateliers flexibles. Dans de tels problèmes, les ressources sont un ensemble de machines M et les tâches sont organisées en travaux. Un travail j correspond à une suite d'activités $[a_{j,1}, \ldots, a_{j,n_j}]$ requise pour construire un objet. Le problème de job-shop consiste, étant donné un ensemble J de travaux, à répartir les tâches des différents travaux en parallèle.

2.2 Ordonnancement avec adversaire

Nous présentons ici les problèmes d'ordonnancement avec adversaires. Deux opposants sont en concurrence : l'*ordonnanceur* veut obtenir un ordonnancement qui satisfait toutes les contraintes (temporelles et de ressources), tandis que l'*adversaire* essaye d'empêcher un tel ordonnancement en détériorant de manière limitée les données du problème. Considérons un exemple avec trois tâches a_1, a_2, a_3 et une ressource r telles que $a_1 \prec a_2$, $d_1 = 1, d_2 = 2, d_3 = 3, c_1 = 3, c_2 = 2$ et $c_3 = 1$. Le projet doit être terminé avant une date $T_{max} = 4$. La solution la plus courte est donnée en figure 2-(i). Supposons que, — *après* qu'un plan a été décidé, — l'adversaire soit capable d'ajouter une unité de besoin en ressources à au plus deux activités, et considérons à nouveau les solutions de la figure 2. La ligne pointillée représente la consommation maximale de ressources en tout point de chaque plan, soumis à toute attaque possible de l'adversaire. On peut voir facilement que la solution la plus courte de la figure 2-(i) peut subir une attaque critique (qui affecterait les tâches a_1 et a_3). La solution linéaire en figure 2-(ii) est robuste en ce qui concerne les attaques de l'adversaire, mais dépasse le temps imparti. Le plan de la figure 2-(iii) termine à temps et est robuste à toute attaque possible.

Le point clé ici est que l'adversaire ne peut utiliser qu'un nombre limité de ressources pour déclencher des attaques, et doit donc résoudre un problème d'optimisation pour identifier la meilleure attaque possible. Cela dépend aussi de s'il agit avant ou après l'établissement du plan. Notons que le fait de limiter le nombre de tâches dont la demande en ressources peut être augmentée fournit simplement un exemple — peut-être le plus simple — de modèle avec adversaire. Nous développons plus amplement ces points dans la partie 4.

2.3 Littérature connexe

Récemment, l'impact de l'incertitude dans les problèmes d'ordonnancement a été considéré dans plusieurs papiers [7, 15, 18, 21]. Le plupart de ces cadres ont pour but de produire des ordonnancements qui "se comportent bien" même si les conditions initiales du problème sont légèrement modifiées, ou si un ennemi ou l'environnement retarde certaines tâches ou utilise certaines ressources pour son usage personnel. Les incertitudes sur les durées des tâches, les consommations de ressources, ou la disponibilité sont modélisées en associant une distribution de probabilités sur les quantités incertaines. Ces probabilités sont prises en compte de manière à produire une solution acceptable d'un point de vue probabiliste [18]. Nous considérons ici un cas légèrement différent : l'adversaire fera de son mieux pour altérer le plan, dans la limite de ses propres capacités, lesquelles sont définies précisément. Ainsi, il n'y a à espérer aucun "comportement moyen" favorable. Nous avons donc besoin d'une analyse du pire cas qui donne une garantie quelque soit le scénario que soit capable de déclencher l'adversaire (voir la notion « conformant fort » ci-dessous).

Les problèmes d'optimisation multi-objectifs ont été étudiés en programmation mathématique sous le nom de *programmation bi-niveau* [4]. Dans ces problèmes, deux décideurs ayant des objectifs différents prennent tour à tour des actions influant sur les mêmes ressources. Il en découle donc un problème d'optimisation dont une des contraintes est elle-même un problème d'optimisation. L'utilisation de la programmation bi-niveau linéaire pour modéliser des ennemis dans un problème d'ordonnancement a été considérée dans [9]. Une modélisation en QCSP du problème de *job-shop* avec des pannes possibles sur les machines a été récemment proposé [16]. En théorie de l'ordonnancement, l'incertitude sur les effets des actions prises (causée par les interactions avec l'environnement) conduit à considérer ce qui pourrait se produire dans le pire des cas. Ces problèmes sont qualifiés de *conformant*. Dans [11], une distinction entre les problèmes conformant faibles (la solution trouvée peut atteindre l'objectif) et forts (la solution trouvée est garantie d'atteindre l'objectif) est faite. Les problèmes conformant faibles supposent que l'adversaire altère le projet par hasard et n'a pas d'intention de nuire, alors que les problèmes conformant forts doivent avoir une solution pour tout comportement malveillant d'un adversaire intelligent.

3 Problèmes quantifiés

Les problèmes de contraintes quantifiées (QCSP) [8] sont une extension du cadre classique des CSP dans lesquels certaines variables peuvent être quantifiées universellement. Dans cette partie, nous définissons les QCSP et les QCSP$^+$, une extension qui permet d'améliorer les possibilités de modélisation et l'efficacité de résolution des QCSP.

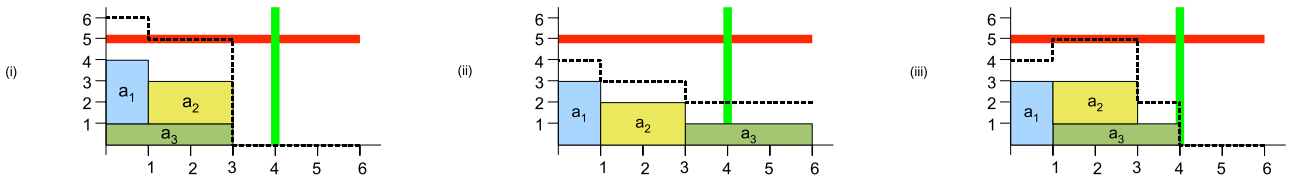

FIG. 2 – Exemple d'ordonnancement avec adversaire.

Un CSP (V, D, C) se compose d'un ensemble $V = \{v_1, \ldots, v_n\}$ de variables définies sur une famille de domaines finis $D = \{d_1, \ldots, d_n\}$ et d'un ensemble $C = \{c_1, \ldots, c_k\}$ de contraintes liant ces variables. Une solution est une affectation de valeurs aux variables du problème telles que toutes les contraintes de C sont satisfaites. L'existence d'une solution à un CSP donné (V, D, C) est équivalent à la satisfaction de la formule suivante :

$$\exists v_1 \in d_1 \ \cdots \ \exists v_n \in d_n \ . \ c_1 \wedge \ldots \wedge c_k$$

Le fait de remplacer certains de ces quantificateurs existentiels par des quantificateurs universels conduit au formalisme des QCSP. Un QCSP est un 5-uple $(V, D, <, q, C)$ où V, D et C sont hérités des CSP, $<$ est un ordre total sur V, et $q : V \rightarrow \exists, \forall$ est une fonction qui associe un quantificateur à chaque variable. Un QCSP $(V, D, <, q, C)$ a alors une solution si, et seulement si la formule suivante est vraie :

$$q(v_1)v_1 \in d_1 \cdots q(v_n)v_n \in d_n \ . \ c_1 \wedge \ldots \wedge c_k$$

Dans la configuration classique des QCSP, toutes les variables sont quantifiées, ce qui donne une formule close.

Les quatre premiers éléments d'un QCSP, c'est à dire $(V, D, <, q)$, définissent le *préfixe* d'un QCSP, qui peut être noté comme une chaîne de *variables quantifiées* $[q(v_1)v_1 \in d_1 \ldots q(v_n)v_n \in d_n]$, ordonnées selon $<$. Une solution d'un tel QCSP n'est plus une simple affectation de toutes les variables de V, car toutes les contraintes doivent maintenant être satisfaites dans plusieurs scénarios, dépendant des valeurs prises par les variables quantifiées universellement. On appelle une telle solution une *stratégie*. Elle est composée d'un ensemble de fonctions s donnant, pour chaque variable existentielle v_i, une valeur en fonction des valeurs prises par toutes les variables universelles qui la précèdent selon $<$. Une stratégie définit un ensemble de scénarios, qui sont les affectations totales compatibles avec la stratégie. Si tous les scénarios satisfont toutes les contraintes de C, la stratégie est dite *gagnante*. Une stratégie gagnante est une solution du QCSP. Le nom stratégie est en référence à la vision intuitive d'un QCSP comme d'un jeu entre le joueur existentiel et le joueur universel.

Décider de l'existence d'une stratégie gagnante d'un QCSP est un problème PSPACE-complet [17], tandis que la résolution d'un CSP est un problème NP-complet. La plupart des solveurs de QCSP utilisent la recherche assistée par une technique de propagation qui est une extension de l'arc-consistance dans le cas quantifié (QAC [8]).

En dépit de leur expressivité théorique, les QCSP souffrent de trois inconvénients majeurs : (i) les mouvements possibles dans la plupart des jeux où deux joueurs s'opposent tour à tour dépend des mouvements précédents (en d'autres termes, les jeux ont des règles... Même les jeux simples comme le morpion, dans lequel il est interdit de jouer deux fois au même endroit). Les QCSP ne fournissent aucun moyen de restreindre proprement ce que le joueur universel est autorisé à faire (pour le joueur existentiel, il suffit d'ajouter les règles comme contraintes additionelles). Les solutions de contournement à ce problème utilisent soit des mécanismes ad-hoc extra-QCSP pour réduire dynamiquement les domaines des variables universelles ou alors une réécriture complexe du modèle[1]. Cette "faille" a été d'abord identifiée pour les QBF, la restriction booléenne des QCSP, dans [2] ; (ii) la propagation ne s'applique qu'aux variables existentielles, ce qui réduit son pouvoir potentiel ; (iii) comme QAC dépend du schéma de quantification des variables, il faut implémenter jusqu'à 2^n propagateurs différents pour une contrainte n-aire.

Les QCSP$^+$ont été présentés dans [6] pour contourner ces problèmes. Il s'agit d'une extension des QCSP dans laquelle chaque quantification peut être arbitrairement restreinte. Dans un QCSP$^+$, un CSP L_i est associé à chaque variable quantifiée v_i de manière à

[1]Une première solution consiste à *réifier* toutes les contraintes et ensuite poster des contraintes disjonctives additionnelles sur les variables réifiées. Une telle réécriture rend le modèle moins lisible, altère sévèrement la propagation, et dans plusieurs cas (y compris celui que nous considérons dans ce papier) ne peut tout simplement pas être utilisée, car la version réifiée des contraintes globales n'est généralement pas implémentée. Une autre technique consiste à utiliser des variables "fantômes" [16]. Cependant, cette technique altère la lisibilité du modèle et requiert certaines propriété sur la structure du problème.

restreindre l'ensemble de ses valeurs possibles. Un tel CSP implique la variable v_i et toutes celles qui la précèdent selon $<$. Formellement, l'association d'une variable quantifiée avec le CSP la restreignant est appelé un *quantificateur restreint* $Q_i = (q_i, v_i, d_i, L_i)$. Un tel objet est noté de la manière suivante : $q_i v_i \in d_i[L_i]$, ce qui se lit "$q_i v_i \in d_i$ tel que L_i est satisfait".

Un QCSP$^+$est une suite de quantificateurs restreints, suivis par un *but*, c'est à dire un CSP qui implique toutes les variables introduites dans les quantificateurs précédents. Par exemple,

$$\exists X \in d_x[L_x]\ \forall Y \in d_y[L_{xy}]\ \exists Z \in d_z[L_{xyz}]\ \ .\ \ C$$

se lit « existe-t-il un $X \in d_x$ tel que L_x et que pour tout $Y \in d_y$ tel que L_{xy} il existe $Z \in d_z$ tel que L_{xyz} et C ? ». Notons que d'un point de vue logique, cette fomule mêle implications et conjonctions en les alternant.

Résoudre un QCSP$^+$ est un problème PSPACE-complet. La preuve est directe en considérant que le problème est équivalent à décider la valeur de vérité de la forme prénexe de la formule logique correspondante, ce qui équivaut à résoudre un QCSP classique.

Les QCSP$^+$ héritent de la propagation classique des CSP, et introduisent un mécanisme original appelé *propagation en cascade* [6], qui, entre autres, permet de réduire les domaines universels. Le solveur QeCode[2], qui est basé sur le solveur de CSP GeCode [19], utilise les QCSP$^+$ comme langage d'entrée. QeCode est le seul solveur quantifié permettant d'utiliser les contraintes globales essentielles à nos modèles.

4 Modèles avec adversaire

Nous considérons deux types de problèmes avec adversaire, en fonction de l'ordre dans lequel l'ordonnanceur et l'adversaire agissent.

Dans le premier cas, un plan (valide) est proposé, et l'adversaire essaye de l'invalider ensuite. Soit s un plan (qui spécifie la date de départ, la durée, et la consommation de ressources de chaque activité), a une attaque, $valid(s)$ le CSP reconnaissant les plans valides, $possible(a, s)$ le modèle de l'adversaire (c'est à dire un CSP qui reconnaît si l'adversaire peut lancer l'attaque a sur le plan s), et $s(a)$ le plan résultant de s après l'attaque a. Nous nous intéressons à décider si :

$$\exists s[valid(s)]\ \forall a[possible(a, s)].\ valid(s(a)) \qquad (1)$$

Si cette formule est vraie, l'ordonnanceur sait que son plan est sûr face à n'importe quelle attaque que l'adversaire puisse effectuer. Dans le cas contraire, l'adversaire sait qu'il est suffisamment puissant pour invalider n'importe quel plan.

Dans le second cas, l'adversaire attaque en premier, et ensuite seulement un plan est établi. Dans ce cas, l'attaque est indépendante du plan, et donc le modèle de l'adversaire est de la forme simplifiée $possible(a)$. Il faut décider si :

$$\forall a[possible(a)]\ \exists s[valid(s)].\ valid(s(a)) \qquad (2)$$

Si cette formule est vraie, l'adversaire sait qu'il est impossible de rendre le problème insoluble. Dans le cas contraire, l'ordonnanceur sait que l'attaquant est suffisamment puissant pour perpétrer une attaque critique.

Ces deux types de problèmes d'ordonnancement avec adversaire peuvent être interprétés comme deux jeux simples à un tour. De tels problèmes se représentent par des problèmes quantifiés dont les préfixes sont respectivement de la forme $\exists\forall$ et $\forall\exists$ et appartiennent donc respectivement aux classes de complexités Σ_2^P et Π_2^P [20]. Il est possible de considérer des cas d'interactions ordonnanceur/adversaire plus généraux, mais nous nous limiterons ici à ces schémas basiques.

Dans les deux schémas sus-cités, un plan est *valide* dans le sens classique, c'est à dire si, et seulement si il satisfait toutes les contraintes temporelles et de ressources. On peut exprimer de telles contraintes en CSP de la manière décrite dans le paragraphe 2.1. Par contre, il n'existe pas de modèle de l'adversaire "standard". Dans le cas le plus simple, le modèle de l'adversaire décrit au paragraphe 2.2 peut être considéré. D'autres exemples de modèles de l'adversaire peuvent, dans certaines limites, allonger les tâches ou imposer des contraintes temporelles additionelles. On peut modéliser d'autre manières de contrecarrer les plans établis à travers une notion d'attaque basée sur le temps : considérons par exemple un adversaire qui peut prolonger ou augmenter la demande en ressources de toutes les activités se déroulant à un instant t^*, mais qui ne peut agir de la sorte qu'à un seul instant. De plus, il suffit de combiner plusieurs de ces modèles simples d'adversaires pour construire des modèles arbitrairement complexes du monde réel.

Ici, nous discutons en détail du modèle QCSP$^+$ du cas (1) en considérant que l'adversaire peut toucher une partie des tâches à ordonner en augmentant leur consommation de ressources par un facteur constant (voir la figure 3).

Soit $A = \{a_1, \ldots, a_n\}$ un ensemble d'activités à planifier et $R = \{r_1, \ldots, r_m\}$ un ensemble de ressources. Chaque activité a_i a une date de début s_i (avec des valeurs dans le domaine ds_i), une durée d_i (que l'adversaire ne peut modifier) et requiert une quantité c_{ij} de ressources r_j. Chaque ressource r_i a une capacité

[2] QeCode est disponible à l'adresse suivante : `http ://www.univ-orleans.fr /lifo/software/qecode`

maximum $rmax_i$. La différence entre les dates de départ de deux activités a_i et a_j telles que $a_i \prec a_j$ est contrainte d'être dans l'intervalle $[min_{ij}..max_{ij}]$.

L'adversaire est modélisé avec deux paramètres $\alpha \in [0..1]$ et $\beta > 0$ qui représentent respectivement la proportion de tâches que l'adversaire peut toucher, et le facteur par lequel la consommation de ressources est augmentée. Par exemple, un adversaire défini par $\alpha = 0.1$ et $\beta = 0.2$ est capable d'augmenter de 20% la consommation en ressources de 10% des tâches. Notons que dans ce modèle simple, le plan élaboré ne joue aucun rôle dans la définition des attaques possibles (c'est à dire que $possible(a, s)$ ne dépend en fait pas de s). Une telle dépendance est en général pertinente (on peut par exemple concevoir qu'un ennemi puisse toucher seulement la première/dernière tâche).

Deux occurrences de la contrainte Cumulative sont présentes dans notre modèle. La première assure que le plan élaboré est réalisable avant l'intervention de l'adversaire. En effet, celui-ci ne pouvant que nuire à son bon déroulement, il est inutile de considérer des plans d'ores et déjà infaisables. La seconde occurrence assure que le plan est toujours réalisable après l'intevention de l'adversaire. Bien que redondante, la première occurrence de Cumulative permet d'accélérer significativement le temps de résolution du problème.

Le modèle décrit en figure 3 se résume donc ainsi : Il existe un ensemble de dates de départ $s_1 \dots s_n$ des tâches qui satisfasse aux contraintes de précédence, tel que les tâches ne dépasseront pas la capacité en ressources en s'exécutant, et tel que pour toute action possible de l'adversaire, chaque action consistant à sélectionner une proportion α de tâches et à leur ajouter un facteur β constant de demande en ressources (ces nouvelles demandes étant représentées par les variables c'_{ij}

5 Expérimentation

Nous avons modifié deux ensembles classiques de problèmes d'ordonnancement multi-ressources (sans adversaire) pour lesquels la date de fin optimale T_{max} a été préalablement calculée[3]. Seules des instances faisables ont été utilisées pour nos tests. Chaque instance a été transformée en un problème avec adversaire de type (1) avec trois paramètres : les paramètres α and β décrits dans le paragraphe précédent, et un troisième paramètre $\gamma > 0$ qui modélise le pourcentage additionnel de temps imparti par rapport à la solution optimale du problème sans adversaire. L'idée est que les plans gagnent en robustesse par rapport aux attaques de l'adversaire en réduisant leur degré de parallélisme, c'est à dire en allongrant leur makespan minimal. En d'autres termes, il nous faut allouer du temps supplémentaire pour améliorer la robustesse.

[3]Ces instances sont décrites dans [13] et sont disponibles au téléchargement sous les noms test-set J10 et test-set J30 sur `http ://www.wiwi.tu-clausthal.de/en/abteilungen/produktion /forschung/schwerpunkte/project-generator/rcpspmax/`

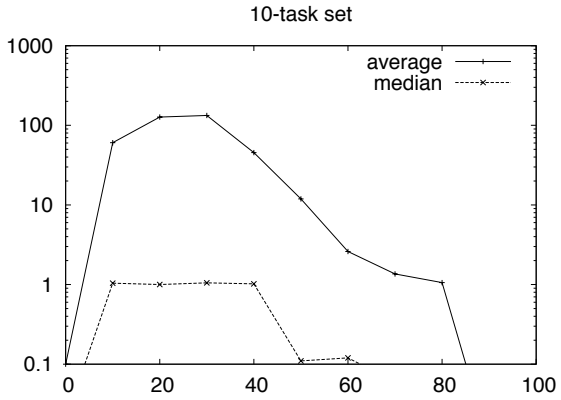

FIG. 4 – Performances moyennes et médianes de QeCode sur un ensemble de problèmes à 10 tâches. L'abscisse représente le pourcentage de tâches touchées par l'adversaire et l'ordonnée donne le temps de résolution. Timeout de 1000 secondes.

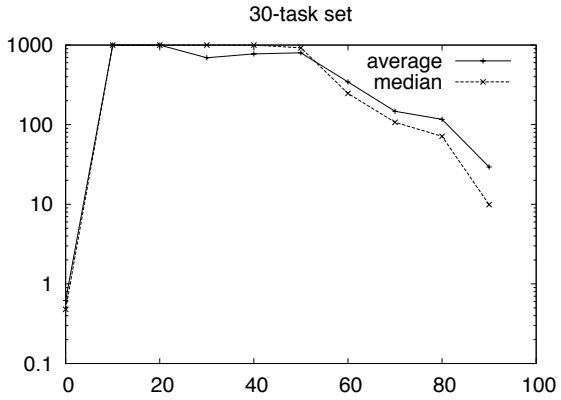

FIG. 5 – Performances moyennes et médianes de QeCode sur un ensemble de problèmes à 10 tâches. Mêmes conditions qu'en figure 4. Timeout de 1000 secondes.

Nous avons exécuté les versions avec adversaire d'instances de problèmes à 10 et 30 tâches. Pour chaque instance initiale, 10 problèmes avec adversaires ont été générés, avec $\alpha = 0, 0.1, \dots, 0.9$ (quand $\alpha = 0$ l'adversaire ne peut affecter aucune tâche, et donc le problème avec adversaire est équivalent au problème initial). Chaque activité affectée par l'adversaire requiert 50% de ressources en plus ($\beta = 0.5$). Le temps maximum alloué est 10% plus long que l'optimum du

$$\exists s_1 \in ds_1, \ldots, s_n \in ds_n,\ s_{end} \in ds_{end}$$
$$[\ \{s_i + d_i \leq s_{end} \mid a_i \in A\} \cup \{s_i + d_i + min_{ij} \leq s_j \mid a_i \prec a_j\} \cup \{s_j \leq s_i + d_i + max_{ij} \mid a_i \prec a_j\} \cup$$
$$\{\text{cumulative}([s_1, \ldots, s_n], [d_1, \ldots, d_n], [c_{1i}, \ldots, c_{ni}], rmax_i) \mid i \in [1..m]\}\ \ ]$$

$$\forall k_1 \in \{0,1\}, \ldots, k_n \in \{0,1\}$$
$$\forall c'_{11} \in [c_{11}..rmax_1], \ldots, c'_{1m} \in [c_{1m}..rmax_m],\ \ \ldots\ \ , \ c'_{n1} \in [c_{n1}..rmax_1], \ldots, c'_{nm} \in [c_{nm}..rmax_m]$$
$$[\ \ \{\textstyle\sum_{i=1..n} k_i = \alpha * n\} \cup \{c'_{ij} = c_{ij} + k_i * (1+\beta) \mid i \in [1..n]\ j \in [1..m]\}\ \ ]$$
$$\{s_{end} \leq T_{max} * (1+\gamma)\} \cup \{\text{cumulative}([s_1, \ldots, s_n], [d_1, \ldots, d_n], [c'_{1i}, \ldots, c'_{ni}], rmax_i) \mid i \in [1..m]\}$$

FIG. 3 – Modèle QCSP$^+$ détaillé d'un problème d'ordonnancement avec adversaire de type (1)

problème initial($\gamma = 0.1$).

Ces tests ont été réalisé sur un dual-opteron à 2.6 GHz équipé de 4 Go de RAM, avec un timeout de 1000 secondes.

Les résultats sont présentés en figures 4 et 5. Comme prévu, un motif facile-difficile-facile se dégage. Pour des petites valeurs de α (adversaire faible), le plan original (avec éventuellement des petites variations) est "vite" reconnu comme une solution. Pour des grandes valeurs de α (40% et plus d'activités attaquées dans le test à 10 tâches), l'adversaires est si puissant qu'aucun plan robuste n'est trouvé. Dans ce cas, le problème est sur-contraint et la propagation montre rapidement l'inconsistance. Les valeurs intermédiaires sont les plus difficiles. La présence d'un adversaire augmente drastiquement le temps de calcul, même quand la version sans adversaire était triviale, tandis que la fraction d'instances positives décroît de 100% à 0%.

Nous recherchons actuellement si la chute des performances est due à la complexité intrinsèque du problème, ou à la nécessité d'élaborer de meilleurs procédés de propagation et d'heuristiques de recherche dans le solveur. Cependant, il est d'ores et déjà notable que le fait d'autoriser des ordonnancements dépassant de 10% le temps de la solution optimale augmente considérablement le nombre de solutions au problème sans adversaire, solutions qui seront potentiellement toutes examinées du point de vue de leur robustesse à l'adversaire.

6 Conclusion

Nous avons montré comment modéliser et résoudre des problèmes d'ordonnancement avec adversaires dans le langage des QCSP$^+$. En plus de l'ordonnancement, il est possible de confronter divers problèmes multi-niveaux du monde réel aux mêmes techniques, comme le montre l'applicabilité de la programmation mathématique bi-niveau [4]. La programmation bi-niveau est souvent limitée au cas linéaire et ne considère que rarement le cas discret. En opposition, notre approche hérite de toute la polyvalence de la programmation par contraintes dans la modélisation de problèmes de décision discrets. De plus, les contraintes définissant la validité d'un ordonnancement (contraintes temporelles et de ressources) ainsi que le modèle de l'adversaire sont simples à écrire en utilisant des contraintes basiques et globales connues de tous les programmeurs par contrainte.

Références

[1] A. Aggoun and N. Beldiceanu. Extending CHIP in Order to Solve Complex Scheduling and Placement Problems. *Mathematical and Computer Modelling*, 17(7) :57–73, April 1993.

[2] C. Ansótegui, C. P. Gomes, and B. Selman. The Achilles' Heel of QBF. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 275–281. AAAI Press AAAI Press / The MIT Press, 2005.

[3] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling*. Operations Research and Management Science. Kluwer Academic Publishers, 2001.

[4] J. F. Bard. *Practical Bilevel Optimization : Algorithms and Applications*, volume 30 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, 1999.

[5] N. Beldiceanu and M. Carlsson. A New Multi-resource Cumulatives Constraint with Negative Heights. In Pascal Van Hentenryck, editor, *CP*, volume 2470 of *Lecture Notes in Computer Science*, pages 63–79. Springer, 2002.

[6] M. Benedetti, A. Lallouet, and J. Vautard. QCSP Made Practical by Virtue of Restricted Quantification. In Manuela Veloso, editor, *International Joint Conference on Artificial Intelligence*, pages 38–43, Hyderabad, India, January 6-12 2007. AAAI Press.

[7] J. Bidot, T. Vidal, P. Laborie, and J. C. Beck. A General Framework for Scheduling in a Stochastic Environment. In Manuela M. Veloso, editor, *IJCAI*, pages 56–61, 2007.

[8] L. Bordeaux and E. Monfroy. Beyond NP : Arc-Consistency for Quantified Constraints. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming*, volume 2470 of *LNCS*, pages 371–386, Ithaca, NY, USA, 2002. Springer.

[9] G. G. Brown, W. M. Carlyle, J. Royset, and R. K. Wood. On the Complexity of Delaying an Adversa-

ry's Project. In *The Next Wave in Computing, Optimization and Decision*, volume 29 of *Operations Research/Computer Science Interfaces Series*, chapter 1. Springer, 2005.

[10] P. Esquirol and P. Lopez. *L'ordonnancement*. Economica, Paris, 1999.

[11] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning*. Morgan Kaufmann, 2004.

[12] R. Graham, E. Lawler, E. Lenstra, and A. Rinnooy Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling : a Survey. *Annals of Discrete Mathematics*, 5 :287–326, 1979.

[13] R. Kolisch, C. Schwindt, and A. Sprecher. Benchmark Instances for Project Scheduling Problems, 1998.

[14] Joseph Leung, Laurie Kelly, and James H. Anderson. *Handbook of Scheduling : Algorithms, Models, and Performance Analysis*. CRC Press, Inc., Boca Raton, FL, USA, 2004.

[15] S.F. Smith N. Policella, A. Oddi and A. Cesta. Generating Robust Partial Order Schedules. In M. Wallace, editor, *Principles and Practice of Constraint Programming*, volume 3258, pages 496–511. Springer, 2004.

[16] P. Nightingale. *Consistency and the Quantified Constraint Satisfaction Problem*. PhD thesis, University of St Andrews, 2007.

[17] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[18] N. Policella. *Scheduling with Uncertainty : A Proactive Approach Using Partial Order Schedules*. PhD thesis, University of Rome "La Sapienza", March 2005.

[19] C. Schulte and G. Tack. Views and Iterators for Generic Constraint Implementations. In Peter van Beek, editor, *CP*, volume 3709 of *Lecture Notes in Computer Science*, pages 817–821. Springer, 2005.

[20] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1) :1–22, 1976.

[21] C.W. Wu, K.N. Brown, and J.C. Beck. Scheduling with Uncertain Durations : Generating Beta-Robust Schedules using Constraint Programming. In *ICAPS 2006 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*, 2006.