Problèmes d'optimisation avec des contraintes quantifiées

Marco Benedetti, Arnaud Lallouet, Jeremie Vautard

Problèmes d'optimisation avec des contraintes quantifiées

Marco Benedetti, Arnaud Lallouet and Jérémie Vautard
Université d'Orléans – LIFO
BP 6759, F-45067 Orléans
{prenom.nom}@univ-orleans.fr

Résumé

Une solution d'un problème de contraintes quantifiées (QCSP) peut être considérée comme une stratégie, qui est une représentation de la manière dont le joueur existentiel réagit au coups du joueur universel. Cependant, ces stratégies ne sont pas toutes équivalentes et certaines peuvent être préférées aux autres. Dans ce papier, nous définissons des problèmes d'optimisations de contraintes quantifiées (QCOP - Quantified constraint Optimization Problem) dans lesquels l'ordre de préférence des sous-stratégies peut être défini sur plusieurs niveaux. Nous montrons que ce formalisme permet de représenter des problèmes de décision hiérarchiques comme les Jeux de Stackelberg ou les problèmes de programmation multi-niveaux.

1 Introduction

La programmation par contraintes quantifiées permet d'exprimer de façon naturelle et concise des problèmes hors de portée des CSP, tels que les jeux à deux joueurs [1, 2, 3] et autres problèmes avec adversaire [4], la vérification de modèles, ou l'ordonnancement robuste par rapport à l'environnement [5].

Résoudre un CSP consiste à trouver une valeur pour chacune de ses variables de manière à ce que toutes les contraintes soient satisfaites. Dans un CSP quantifié (QCSP), une variable peut être quantifiée de manière universelle sur son domaine, ce qui permet d'exprimer une incertitude sur la valeur que peut prendre cette variable, ce qui permet de modéliser le comportement possible d'un adversaire, ou une incertitude sur l'environnement. Un QCSP est vrai si on peut trouver des valeurs pour les variables existentielles restantes qui soient consistantes avec toutes les valeurs des universelles. Ainsi, la notion de solution d'un QCSP est une famille de fonctions de Skolem appelée *stratégie* qui associe une valeur à chaque variable existentielle en fonction des variables universelles précédentes. Une telle stratégie n'est pas un objet compact : sa taille est dans le cas général exponentielle en le nombre de variables. Un QCSP est vrai s'il possède au moins une stratégie *gagnante*, c'est à dire dans laquelle les valeurs données aux variables existentielles pour chaque affectation possible des variables universelles satisfont toutes les contraintes.

Toutes les stratégies gagnantes sont-elles équivalentes ? Dans la modélisation d'un jeu, il pourrait être intéressant de trouver la stratégie permettant de gagner le plus rapidement possible. Dans beaucoup de cas. les stratégies peuvent être évaluées et ordonnées selon une préférence donnée. Le moyen le plus facile d'exprimer une préférence sur les solutions est de définir une fonction de l'ensemble des stratégies vers un ensemble ordonné et de choisir une stratégie qui optimise cette valeur. C'est précisément ce que nous faisons ici. Dans les CSP, on peut exprimer une telle optimisation en demandant une solution qui maximise ou minimise telle variable du problème. Cependant les stratégies sont des objets complexes, et nous proposons, pour exprimer une telle fonction d'optimisation, de définir un langage appelé QCOP$^+$, basé que les QCSP$^+$ [3]. Ce langage suit la structure récursive de la formule d'un QCSP

$$\begin{array}{l}\exists X \in D_X \,.\, [C_1(X)] \\ \quad \forall Y \in D_Y \,.\, [C_2(X,Y)] \\ \quad\quad \exists Z \in D_Z \,.\, [C_3(X,Y,Z)] \\ \quad\quad\quad C(X,Y,Z) \\ \quad\quad \min(Z) \\ \quad s : sum(Z) \\ \max(s)\end{array}$$

FIG. 1 – Exemple de QCOP$^+$

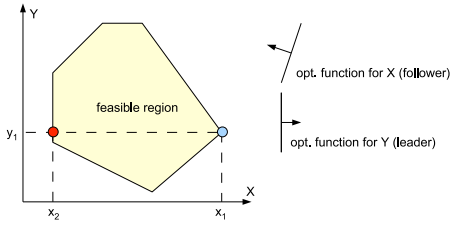

FIG. 2 – Optimum pour le meneur seul, et réponse du suiveur

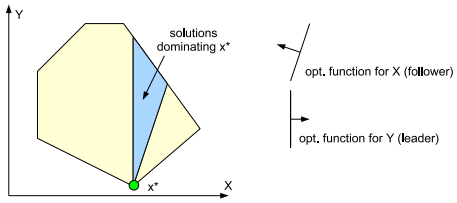

FIG. 3 – Équilibre optimal

et permet d'optimiser sur les variables existentielles et de calculer des agrégats au niveau des variables universelles. La figure 1 donne un exemple d'un problème d'optimisations entrelacées en QCOP$^+$ (chaque optimisation et calcul d'agrégat est associé au quantificateur ayant la même indentation). Le but est de trouver une valeur X qui maximise la valeur de l'agrégat s calculé à partir de la sous-stratégie. Au niveau inférieur suivant, X est instanciée à la valeur choisie et un aggrégat est calculé en sommant les valeurs de Z retournées pour chaque valeur de Y. A un niveau encore inférieur, la valeur de Z retournée est la plus petite valeur possible telle que $C(X, Y, Z)$ est vraie. Les conditions C_1 à C_3 viennent restreindre les valeurs possible d'une variable de manière dynamique. Ainsi, la somme calculée dans s est la somme des valeurs minimales de Z. Cet agrégat est alors évalué pour toutes les valeurs de X, et on retourne une stratégie affectant une valeur à X qui maximise s. Le nombre de niveaux possibles n'est pas limité.

En regardant dans la littérature sur les QCSP, nous n'avons pas trouvé de notion générale d'optimisation. Cependant, des problèmes de ce genre sont étudiés depuis les années 1970 en programmation mathématique sous le nom de problèmes de programmation *bi-niveaux* ou *multi niveaux*. [6]. Les programmes bi-niveaux sont utilisés pour résoudre des problèmes de décision sous la forme de jeux de Stackelberg, qui est un modèle d'oligopole en théorie des jeux [7]. Dans ces problèmes sont représentés deux acteurs agissant de manière séquentielle, mais n'ayant aucun pouvoir l'un sur l'autre. Le premier à décider est appelé le meneur, et le second (appelé le suiveur) prend acte de cette décision et adapte sa propre décision en fonction. Le problème est que le meneur et le suiveur ont des fonctions d'objectif différentes, qui ne sont pas forcément concordantes ni opposées. Par exemple, le meneur peut être une agence gouvernementale qui partage des fonds disponibles entre plusieurs entités dont chacune a la liberté d'utiliser sa dotation comme elle l'entend. L'exemple suivant, pris dans [8], montre que des objectifs en conflit peuvent amener à un équilibre non-optimal. Dans la situation exposée en figure 2, le choix du couple (x_1, y_1) qui serait optimal sans le suiveur devient considérablement sous-optimal quand ce dernier entre en jeu, menant à la solution (x_2, y_1). L'équilibre x^* est montré en figure 3, où l'on peut noter qu'il existe des solution strictement dominantes et pour le meneur, et pour le suiveur, qui ne peuvent jamais être atteintes sans consensus. Nous nous référons à [9] pour une étude extensive sur la programmation bi-niveau. Le terme "programmation multi-niveaux" s'applique dès lors qu'il y a plus de deux niveaux de décisions hiérarchiques.

Nous avons intégré le formalisme des QCOP$^+$ dans le solveur QeCode [10] basé sur Gecode [11], ainsi qu'une extraction simple de la stratégie gagnante, ce qui est aussi utile pour les QCSP$^+$ classiques : dans beaucoup de situations, une simple réponse "booléenne" ne suffit pas étant donné que l'utilisateur désire obtenir les valeurs des variables (comme pour les CSP). L'extraction des stratégies a été présentée dans [12] dans le cadre des QBF. Ce papier présente les QCSP, les QCSP$^+$, et les QCOP$^+$ ainsi que leur résolution. Nous y présentons l'algorithme de recherche et en étudions des premières variantes de branch-and-bound. Enfin, nous donnons quelques exemples de modélisation de problèmes bi-niveaux.

2 QCSP

Notations. soit V un ensemble de variables et $D = (D_X)_{X \in V}$ la famille de leurs domaines. On rappelle qu'une famille est une fonction d'un ensemble indicé dans un ensemble. Pour un sous-ensemble $W \subseteq V$, on note D^W l'ensemble des n-uplets de W, c'est à dire $\Pi_{X \in W} D_X$. La projection d'un n-uplet (ou d'un ensemble de n-uplets) sur une variable (ou un ensemble de variables) est notée $|$. Par exemple, pour $t \in D^V$, $t|_W = (t_x)_{X \in W}$ et pour $E \subseteq D^V$, $E|_W = \{t|_W \mid t \in E\}$. Pour $W, U \subseteq V$, la jointure de $A \subseteq D^W$ et $B \subseteq D^U$ est $A \bowtie B = \{t \in D^{A \cup B} \mid t|_W \in A \ \wedge \ t|_U \in B\}$. Une séquence est une famille indexée par un préfixe de $\mathbb{N}$. On note $|$ le constructeur de séquence et $[]$ la séquence vide. On utilise la notation $a\,?b\,:c$ pour exprimer *si a alors b sinon c*.

Contraintes et CSP. Une *contrainte* $c = (W, T)$ est un couple composé d'un sous-ensemble $W \subseteq V$ de variables et d'une relation $T \subseteq D^W$ (W et T sont aussi respectivement notés $var(c)$ et $sol(c)$). Une contrainte vide (telle que $sol(c) = \emptyset$) est *fausse* et une contrainte pleine est telle que $sol(c) = D^W$. Quand $W = \emptyset$, seules ces deux contraintes existent : $(\emptyset, \emptyset)$ qui a pour valeur *faux* et $(\emptyset, ())$ qui a pour valeur *vrai*.
Un *problème de satisfaction de contraintes* (CSP) est un ensemble de contraintes. On note $var(C) = \bigcup_{c \in C} var(c)$ l'ensemble de ses variables et $sol(C) = \bowtie_{c \in C} sol(c)$ l'ensemble de ses solutions. Le CSP vide – ne contenant aucune contrainte– est *vrai* et est noté $\top$, tandis que tout CSP contenant une contrainte fausse est lui-même *faux* et noté $\bot$.

Préfixe et QCSP. Un *ensemble de variables quantifié*, ou *qset* est un couple (q, W) où $q \in \{\exists, \forall\}$ est un quantificateur et $W \subseteq V$.

Definition 1 (Préfixe) *Un A* préfixe P *est une séquence de qsets* $[(q_0, W_0), \ldots, (q_{n-1}, W_{n-1})]$ *tel que* $i \neq j \Rightarrow W_i \cap W_j = \emptyset$.

On note $P|_W$ la restriction du préfixe P aux variables d'un ensemble W. Une variable X est *déclarée* dans un qset W_i si $X \in W_i$. Un QCSP est défini en ajoutant un CSP à un préfixe :

Definition 2 (QCSP) *Un* CSP quantifié*, ou* QCSP *est un couple* (P, G) *où* P *est un préfixe et* G *un CSP appelé* goal.

Soit $P = [(q_0, W_0), \ldots, (q_{n-1}, W_{n-1})]$ un préfixe. Définissons les notations suivantes :

Premièrement, soit $range(P) = [0..n]$. Pour tout i de $range(P)$, soit $var_i(P) = W_i$ l'ensemble des variables déclarées à l'indice i, soit $before_i(P) = \bigcup_{j \leq i} var_j(P)$ (resp. $after_i(P) = before_n(P) \backslash before_i(P)$) l'ensemble des variables définies avant (resp. après) l'indice i.

Nous devons aussi pouvoir accéder à l'indice du prochain bloc universel $nu_i(P)$ situé après l'indice i. Définissons $nu_i(P) = min_{j>i}\{j \mid q_j = \forall\}$ si un tel indice existe, n sinon. Ces notations sont naturellement et directement étendues aux QCSP $Q = (P, G)$. De plus, on a $prefix(Q) = P$ et $goal(Q) = G$. Le QCSP est dit *clos* si $var(G) = before_n(Q)$, c'est-à-dire si toutes les variables mentionnées dans le goal sont explicitement quantifiées. Par la suite, nous considérerons uniquement des QCSP clos.

Example 3 (QCSP) *La formule :*

$$\exists X \in \{0,1\},\ \forall Y \in \{0,1\},\ \exists Z \in \{1,2\}\ .\ X + Y = Z$$

est représentée par le QCSP suivant (les domaines des variables n'étant pas mentionnés) :

$$Q = ([(\exists, X), (\forall, Y), (\exists, Z)], \{X + Y = Z\})$$

Ainsi, $prefix(Q) = [(\exists, X), (\forall, Y), (\exists, Z)]$, $goal(Q) = \{X + Y = Z\}$, $range(Q) = [1..3]$, $var_1(Q) = \{X\}$, $before_2(Q) = \{X, Y\}$, $after_2(Q) = \{Z\}$. □

Stratégie et scénario. La notion de solution d'un QCSP ne peut pas être une simple affectation de toutes ses variables comme pour un CSP. Elle doit être un ensemble d'affectations compatible avec la quantification universelle de certaines des variables. Intuitivement, une solution, appelée *stratégie*, contient la façon dont le joueur existentiel réagit à tous les coups possibles du joueur universel. Il est intéressant de noter qu'une stratégie est un objet syntactique indépendant de toute notion de validité : il s'agit juste d'un moyen possible de jouer au jeu comme s'il n'y avait pas de règle. Par conséquent, elle peut être définie uniquement sur un préfixe. Dans [13], une stratégie était définie comme une famille de fonctions (de Skolem) donnant chacune une valeur à une variable existentielle en fonction des universelles précédentes. Pour exposer cette notion d'un point de vue ensembliste, nous définissons plutôt une stratégie en extension, comme étant un ensemble de n-uplets. chacun de ces n-uplets est un *scénario*, c'est-à-dire un déroulement possible du jeu. La définition (inductive) de l'ensemble des stratégies pour un préfixe donné est la suivante :

Definition 4 (ensemble des stratégies) *L'*ensemble des stratégies $Strat(P)$ *d'un préfixe* $P = [(q_0, W_0), \ldots, (q_{n-1}, W_{n-1})]$ *est définie inductivement comme suit :*

- $Strat([]) = \emptyset$
- $Strat([(\exists, W) \mid P']) = \{t \bowtie s' \mid t \in D^W \wedge s' \in Strat(P')\}$
- $Strat([(\forall, W)|P']) = \{ \bigcup \alpha(D^W) \mid \alpha \in \Pi_{t \in D^W}(\{t \bowtie s' \mid s' \in Strat(P')\}) \}$

L'ensemble des stratégies pour un préfixe commençant par une variable universelle se définit ainsi : on construit, pour un n-uplet $t \in D^W$, l'ensemble $\{ t \bowtie s' \mid s' \in Strat(P') \}$ de toutes les stratégies commençant par t, et on prend le produit cartésien $\Pi_{t \in D^W}(\{ t \bowtie s' \mid s' \in Strat(P') \})$ de tous ces ensembles. Chaque n-uplet α de ce produit cartésien est aussi une fonction associant à chaque n-uplet de D^W une stratégie, qui est un ensemble de n-uplets. L'union des stratégies de l'ensemble image $\alpha(D^W)$ de cette fonction est une nouvelle stratégie contenant une sous-stratégie pour chaque $t \in D^W$. L'ensemble des stratégies pour le préfixe est l'ensemble de toutes les stratégies construites par tous les n-uplets α.

Sémantique d'un QCSP. Une stratégie est *gagnante* si tous ses scénarios satisfont le goal :

Definition 5 (Stratégie gagnante d'un QCSP) *Une stratégie s est une* stratégie gagnante *si, et seulement si $s|_{var(G)} \subseteq sol(G)$.*

On note $\text{WIN}(Q)$ l'ensemble de toutes les stratégies gagnante de Q.

Definition 6 (Sémantique d'un QCSP) *La* sémantique $[\![Q]\!]$ *d'un QCSP Q est :*

$$[\![Q]\!] = Win(Q)$$

Cette notion de solution généralise exactement la notion classique d'un CSP : un QCSP est vrai si et seulement si il admet une stratégie gagnante. D'autres notions plus faibles ont été proposées. [13] a utilisé la notion d'*outcome*, qui est l'ensemble des scénarios de toutes les stratégies gagnantes, comme notion de solution d'un QCSP pour modéliser le filtrage.

Example 7 *Considérons les QCSP suivants :*

$$\begin{array}{lll} Q_1: & \forall x \in \{0,1\}, \exists y \in \{1\}, \exists z \in \{0,1\}. & x \vee y = z \\ Q_2: & \exists x \in \{0,1\}, \forall y \in \{1\}, \forall z \in \{1\}. & x \vee y = z \\ Q_3: & \exists x \in \{0,1\}, \forall y \in \{0,1\}, \exists z \in \{1\}. & x \vee y = z \end{array}$$

On a alors :

$$\begin{array}{lcl} [\![Q_1]\!] & = & \{\ \{(0,1,1),(1,1,1)\}\ \} \\ [\![Q_2]\!] & = & \{\ \{(0,1,1)\},\ \{(1,1,1)\}\ \} \\ [\![Q_3]\!] & = & \{\ \{(1,0,1),(1,1,1)\}\ \} \end{array}$$

QCSP$^+$. Introduire la quantification restreints dans les QCSP passe par une modification de la nature du préfixe. en plus d'un quantificateur et d'un ensemble de variables, on ajoute un CSP dont les solutions définissent les valeurs autorisées pour les variables du qset courant. Les QCSP$^+$ ont été définis dans [3], essentiellement pour pallier à des problèmes de modélisation. Un *ensemble de variables quantifiées de manière restreinte*, ou *rqset* est un triplet (q, W, C) où $q \in \{\exists, \forall\}$ est un quantificateur, $W \subseteq V$ et C un CSP. Le but est de restreindre les valeurs possibles des variables de W à celles qui satisfont le CSP C. Nous étendons la notion de préfixe à ces rqsets : en particulier, $i \neq j \Rightarrow W_i \cap W_j = \emptyset$ doit toujours être vérifié.

Definition 8 (QCSP$^+$) *Un QCSP$^+$ est un couple $Q = (P, G)$ où P est un préfixe de rqsets tel que $var(C_i) \cap after_i(Q) = \emptyset$ et G est un CSP goal.*

Un QCSP$^+$ $Q = (P, G)$ est clos si $\forall i \in range(P), var(C_i) \subseteq before_i(Q)$ et $var(G) \subseteq before_n(Q)$. On remarque aisément qu'un QCSP standard est un QCSP$^+$ où $\forall i \in range(P), C_i = \emptyset$. La définition d'une stratégie d'un QCSP$^+$ est la même que pour un QCSP. Par contre, la notion de stratégie *gagnante* est différente : une stratégie gagnante est une stratégie pour laquelle tous les coups possibles du joueur universel mènent à un scénario gagnant. Comme dans les QCSP classiques, cela peut arriver quand toutes les contraintes du goal et des restrictions sont satisfaites. Mais cela peut aussi arriver quand la partie gauche d'une implication est fausse : ce scénario est alors valide quelles que soient les affectations des variables situées avec le rqset en question.

L'ensemble des stratégies gagnantes d'un QCSP$^+$ peut aussi être défini récursivement de la manière suivante :

Definition 9 (Stratégies gagnantes d'un QCSP$^+$) *Soit Q un QCSP$^+$. L'*ensemble des stratégies gagnantes $\text{WIN}(Q)$ *est défini par :*

- $\text{WIN}((\,[\,], G)) = sol(G)$
- $\text{WIN}((\,[(\exists, W, C)|P'], G)) = \{t \bowtie s \mid t \in D^W \wedge t|_{var(C)} \in sol(C) \wedge s \in \text{WIN}(P', G)\}$
- $\text{WIN}((\,[(\forall, W, C)|P'], G)) = \{\ \bigcup \alpha(D^W) \mid \alpha \in \Pi_{t \in D^W}(\ \{\ t \bowtie s \mid t|_{var(C)} \in sol(C)\ ?\ s \in \text{WIN}(P', G) : s \in \text{STRAT}(P', G)\ \}\ )\ \}$

Cette définition est analogue à la définition de l'ensemble des stratégies pour un préfixe, mais les sous-stratégies gagnantes ne sont pas les seules à être utiles, étant donné qu'une stratégie peut être gagnante à un niveau universel si elle contredit la restriction du niveau en question. Dès lors, n'importe quelle sous-stratégie, qu'elle soit gagnante ou non, peut être attachée.

La définition de la sémantique d'un QCSP s'applique aussi aux QCSP$^+$. Une méthode de propagation dans les QCSP+, appelée propagation en cascade, est décrite dans [3].

3 Optimisation

Les QCOP$^+$ sont créés à partir des QCSP$^+$ en ajoutant des fonctions de préférence et des agrégats sur les rqsets. Soit $\mathcal{A}$ un ensemble de noms d'agrégats et $\mathcal{F}$ un ensemble de fonctions d'agrégats. On définit une fonction d'agrégat f comme étant une fonction associant une valeur à un multi-ensemble, et dotée d'un élément neutre 0_f indiquant la valeur de $f(\{\!\{\}\!\})$. *somme, produit, moyenne, écart-type, médiane, cardinalité*,etc. sont des exemples de telles fonctions. Un *agrégat* est un atome de la forme $a : f(X)$ où $a \in \mathcal{A}, f \in \mathcal{F}$ et $X \in V \cup \mathcal{A}$. On appelle $names(A)$ l'ensemble des noms d'agrégats associé à un ensemble d'agrégats A. Une *condition d'optimisation* est un atome de la forme $\min(X), \max(X)$ où $X \in V \cup \mathcal{A}$, ou l'atome *any*. Un atome $\min(X)$ indique que l'on s'intéresse aux stratégies qui minimisent X et pas aux autres, tandis que

any indique l'absence de préférence sur les stratégies retournées. $max(X)$ est équivalent à $min(-X)$.

Definition 10 (Orqset) *Un* $\exists$-orqset *est un 4-uple* $(\exists, W, C, o)$ *où* $(\exists, W, C)$ *est un rqset et* o *une condition d'optimisation. Un* $\forall$-orqset *est un 4-uple* $(\forall, W, C, A)$ *où* $(\exists, W, C)$ *est un rqset et* A *un ensemble d'agrégats. Un* orqset *est soit un* $\exists$*-orqset soit un* $\forall$*-orqset.*

Nous étendons la notion de préfixe, ainsi que toutes les notations associées, à une séquence d'orqsets. Une restriction sur les variables possibles d'une condition d'optimisation ou d'un agrégat peut cependant apparaître : une variable à optimiser doit en effet avoir une unique valeur dans la stratégie courante. C'est effectivement le cas si il n'y a pas de $\forall$-orqset entre la condition d'optimisation et la définition de la variable à optimiser. Il est cependant possible d'optimiser sur un agrégat défini exactement au prochain bloc universel, étant donné que là encore, nous obtiendrons une valeur unique. Il en est de même pour une variable à agréger : celle-ci peut appartenir à l'ensemble des variables de n'importe quel bloc existentiel situé entre le bloc où l'agrégat est déclaré et le bloc universel suivant.

Considérons par exemple la séquence d'orqset suivante : $(\exists, W, C, o)$, $(\exists, W_2, C_2, o_2)$, $(\forall, W_3, C_3, A)$. La condition d'optimisation o peut porter sur les variables de W ou de W_2, ou encore sur une valeur d'agrégat obtenu à partir d'un élément de A. en effet, dans une sous-stratégie correspondante à cette séquence, ces variables ont une unique valeur. en revanche, o ne peut pas porter sur une des variables de W_3 étant donné que celles-ci prennent toutes les valeurs possibles dans la sous-stratégie.

Definition 11 (QCOP et QCOP$^+$) *Un QCOP*$^+$ *est un couple* (P, G) *où* G *est un CSP et* $P = [orq_0, \ldots, orq_{n-1}]$ *un préfixe d'orqsets tel que* $\forall i \in range(P)$*, avec* $k = nu_i(P)$ *:*

- *si* $orq_i = (\exists, W, C, o)$ *avec* $o = min(X)$ *ou* $o = max(X)$*, alors on doit avoir* $X \in before_{k-1}(P) \cup (k < n \ ? \ names(A_k) : \emptyset)$
- *si* $orq_i = (\forall, W, C, A)$*, alors pour tout* $a : f(X)$ *in* A*, on doit avoir* $X \in before_{k-1}(P) \cup (k < n \ ? \ names(A_k) : \emptyset)$

Un QCOP est un QCOP$^+$ *dans lequel aucun orqset ne possède de restriction.*

La sémantique d'un QCOP$^+$ est définie comme un ensemble de stratégies incluant le calcul des agrégats et respectant les conditions d'optimisation. Définissons pour commencer la fonction *val* qui calcule la valeur d'un agrégat $a{:}f(X)$ pour une stratégie s donnée. On a $val(a{:}f(X), s) = f(\{\!\{t|_X \mid t \in s\}\!\})$.

Definition 12 (Sémantique d'un QCOP$^+$) *La sémantique d'un QCOP*$^+$ *est un ensemble de stratégies tel que :*

- $\text{WIN}(([], G)) = sol(G)$
- $\text{WIN}(([(\exists, W, C, any)|P'], G)) = \text{WIN}(([(\exists, W, C)|P'], G))$
- $\text{WIN}(([(\exists, W, C, min(X))|P'], G)) = \{s \in \text{WIN}(([(\exists, W, C)|P'], G)) \mid s|_X = \min_{s' \in \text{WIN}(([(\exists, W, C)|P'], G))}(s'|_X)\}$
- $\text{WIN}(([(\forall, W, C, A)|P'], G)) = \{(val(a : f(X), s))_{a \in names(A)} \bowtie s \mid s \in \text{WIN}(([(\forall, W, C, A)|P'], G))\}$

Une fois les agrégats calculés, leurs valeurs sont attachés aux scénarios de la stratégie et ils apparaissent comme s'ils étaient des variables existentielles du niveau suivant. Tout comme un CSP peut avoir plusieurs solutions optimales, un QCOP$^+$ peut avoir plusieurs stratégies optimales. Cela peut se produire non seulement quand on utilise *any*, mais aussi quand plusieurs stratégies ont la même valeur optimale. Les sous-stratégies (ainsi que leurs valeurs optimales) peuvent varier énormément d'une stratégie optimale à l'autre. Cependant, l'algorithme de recherche décrit dans la section suivante retourne une seule de ces stratégies optimales.

4 Algorithmes

Cette section présente un algorithme de recherche évaluant les QCOP$^+$ et esquisse une amélioration basée sur le principe du branch-and-bound. Cet algorithme de recherche a été implémenté dans le solveur QeCode [10], basé sur Gecode [11]. Cette technique de résolution est basée sur la procédure de recherche des QCSP$^+$ qui explore la structure de quantification du problème de manière récursive. Un mécanisme d'extraction de la stratégie et son stockage sous forme d'arbre a été ajouté. Cette dernière fonctionnalité améliore aussi la réponse donnée à un QCSP$^+$, étant donné que l'utilisateur ne s'intéresse généralement pas au problème de décision lui même mais aussi à la manière précise de "gagner au jeu". Une représentation explicite des stratégies – que [12] appelle certificats – possède de nombreuses application, la première étant de pouvoir vérifier la solution d'une manière indépendante du solveur. Actuellement, la stratégie est stockée de manière non compressée, ce qui peut constituer une limite à la taille des instances pouvant être résolues.

La procédure de recherche principale se compose de deux fonctions d'évaluation mutuellement récursives, l'une traitant les $\exists$-orqset et l'autre dédiée aux $\forall$-orqset. Toutes deux retournent une stratégie sous forme d'un arbre qui peut être soit l'arbre vide *null* soit *tree(a,B)* où a est un n-uplet et B un ensemble

d'arbres. La figure 4 montre ces trois algorithmes. Pour un ∃-orqset, la fonction garde en mémoire la meilleurs stratégie rencontrée jusque là (*BEST_STR*) et la retourne, ou renvoie null si le sous-problème est faux. Toutes les stratégies sont successivement explorées et leur valeur de X comparées. Les conditions max et *any* sont traitées de manière similaire.

Procedure Solve $([o|P'], G)$
  if o = orqset existentiel **then**
    return Solve_e $([o|P'], G)$
  else
    return Solve_u $([o|P'], G)$
  end if

Procedure Solve_e($[(\exists, W, C, \min(X))|P'], G$)
  BEST_STR := null
  BEST_Xvalue := $+\infty$
  for all $t \in D^W$ t.q. t solution de C **do**
    CUR_STR := Solve($(P', G)[W \leftarrow t]$)
    if CUR_STR $\neq$ null **then**
      CUR_Xvalue := CUR_STR$|_X$
      if CUR_Xvalue < BEST_Xvalue **then**
        BEST_STR := CUR_STR
        BEST_Xvalue := CUR_Xvalue
      end if
    end if
  end for
  return tree(t,{ BEST_STR })

Procedure Solve_u $([(\forall, W, C, A)|P'], G)$
  for all $a{:}f(X) \in A$ **do**
    VAL_a := $\emptyset$
  end for
  STR := $\emptyset$
  for all $t \in D^W$ t.q. t solution de C **do**
    CUR_STR := Solve($(P', G)[W \leftarrow t]$)
    if CUR_STR = null **then**
      return null
    else
      for all $a{:}f(X) \in A$ **do**
        VAL_a := VAL_a $\cup$ $\{\!\{$ CUR_STR$|_X$ $\}\!\}$
      end for
      STR := STR $\bigcup$ CUR_STR
    end if
  end for
  return tree((f(VAL_a))$_{a:f(X)\in A}$, STR)

FIG. 4 – Procédure de recherche.

La forme la plus simple de branch-and-bound applicable à cette procédure de recherche consiste à simplement poster la contrainte $X < BEST\_Xvalue$ (resp. >) aux branches restantes à explorer du problème de minimisation (resp. maximisation). Il s'agit d'une adaptation directe de l'algorithme de [14] pour lequel les bornes inférieure/supérieure sont fixées par la condition d'optimisation et associées à sa variable d'optimisation. Quand une solution est trouvée, la partie de l'algorithme dédiée aux ∀-orqsets évalue les sous-stratégies découlant de chaque n-uplet valide pour cet orqset, calcule les agrégats et finalement retourne l'ensemble *STR* de ces sous-stratégies.

Branch and Bound. Cet algorithme de Branch-and-Bound est incorrect dans le cas où plusieurs conditions d'optimisations se chevauchent, c'est-à-dire si il existe deux orqsets $orq_i = (\exists, W_i, C_i, min(X))$ avec $X \in W_k$ et $orq_j = (\exists, W_j, C_j, min(Y))$ avec $Y \in W_l$ tels que $i < j < k$.

$\exists X \in D_X$
  $\exists Y \in D_Y$
    $\exists A \in D_A$
      $\exists B \in D_B$
        ...
      any
    any
  min(B)
min(A)

FIG. 5 – Cas où le B&B est incorrect

Example 13 *Un exemple de problème où le branch-and-bound est incorrect est donnéen figure 5. Supposons qu'il existe trois stratégies* $s0 = \{(X_0, Y_0, A_0, B_0)\}$, $s_1 = \{(X_1, Y_1, A_1, B_1)\}$ *et* $s_2 = \{(X_1, Y_2, A_2, B_2)\}$ *telles que* $A_1 > A_0$, $A_2 < A_0$ *et* $B_1 < B_2$. *Une fois* s_0 *trouvée, la contrainte* $A < A_0$ *est ajoutée à la recherche des stratégies suivantes. Et donc,* s_1 *n'est plus considérée, et on trouve* s_2 *dont la valeur* A_2 *pour* A *est meilleure, ce qui en fait la stratégie optimale retournée. Or, sans l'application du branch-and-bound, la condition d'optimisation au niveau de* Y *aurait retourné la stratégie* s_1, *sa valeur de* B *étant meilleure que celle de* s_2, *et donc la valeur* A_1 *aurait été retournée au niveau supérieur, et la meilleure stratégie au niveau supérieur aurait donc bien été* s_0.

Proposition 14 *Le branch-and-bound est correct aux niveaux où des conditions d'optimisations ne se chevauchent pas.*

Idée de la preuve En utilisant les mêmes notations que ci-dessus, les conditions d'optimisations ne se chevauchent pas si $k \leq j$. Dès lors, n'importe quelle branche coupée par le branch-and-bound le sera avant que l'on atteigne le niveau de Y, et donc, seules des stratégies moins bonnes pour X seront supprimées. □

5 Examples

Dans cette section, nous donnons plusieurs exemples de modélisation de problèmes utilisant l'optimisation,

allant des exemples jouets à des problèmes réels pris dans la littérature sur la programmation bi-niveaux. Nous utilisons une syntaxe dans laquelle les agrégats et les conditions d'optimisations apparaissent à la fin, de manière à rendre les problèmes plus lisibles. Par exemple, voici un QCOP$^+$ qui retourne une stratégie dans laquelle $X = 0$ si la somme des indices impairs du tableau A est inférieur à la somme de ses indices pairs et $X = 1$ sinon, présenté de manière formelle, puis dans cette syntaxe :

$$
\begin{array}{l}
(\ [\ (\exists, \{X\}, \emptyset, min(s)), \\
\quad (\forall, \{i\}, \{i \bmod 2 = X\}, \{s : sum(Z)\}), \\
\quad\quad (\exists, \{Z\}, \emptyset, any) \ ], \\
\quad\quad\quad \{Z = A[i]\} \ )
\end{array}
$$

const A[0..9]
∃ X in 0..1
| ∀ i in 0..9 [i mod 2 = X]
| | ∃ Z in 0..+oo
| | | Z = A[i]
| | any
| s :sum(Z)
min(s)

Ordonnancement avec adversaire et minimax. Souvent, les objectifs de deux agents sont strictement opposés. Cette situation peut être résolue par un algorithme minimax classique. Dans ce cas, le branch-and-bound est équivalent au filtrage alpha-beta. Nus illustrons ce cas avec un extension de l'exemple de l'ordonnancement avec adversaire présenté dans [4]. Ce problème implique deux opposants : l'*ordonnanceur* qui cherche à établir un plan respectant des contraintes données (ressources disponibles et temps imparti), tandis qu'un *adversaire* cherche à empêcher l'établissement d'un tel plan en modifiant (dans certaines limites) les données du problème original. Dans la version en QCSP$^+$ de ce problème, l'ordonnanceur cherchait à établir un plan se terminant avant une date limite donnée. Il est maintenant possible de chercher à minimiser ce temps pour une attaque donnée, tandis que l'adversaire cherche à maximiser ce même temps. Considérons par exemple trois tâches a_1, a_2, a_3 et une ressource r. Chaque tâche a_i a une date de début s_i, une durée d_i et demande c_i unités de la ressource r, dont la capacité maximale est 5. L'ordre de précédence des tâches est $a_1 \prec a_2$, et les données du problème sont $d_1 = 1, d_2 = 2, d_3 = 3, c_1 = 3, c_2 = 2$ et $c_3 = 1$. L'adversaire est capable d'augmenter la consommation de ressource d'au plus deux tâches d'une unité. Nous ajoutons une tâche factice *end* qui servira à minimiser la durée totale du plan. Le QCOP$^+$ est le suivant :

const d[1..3], c[1..n]
∃ k1 ∈ 0,1, k2 ∈ 0,1, k3 ∈ 0,1
| [k1+k2+k3 =< 2]
| ∃ S1 ∈ D1, S2 ∈ D2, S3 ∈ D3, Send ∈ Dend,
| | c'1 ∈ Dc1, c'2 ∈ Dc2, c'3 ∈ Dc3
| | [S1+d1 =< Send, S2+d2 =< Send, S3+d3 =< Send,
| | S1+D1 =< S2, c'1=c1+k1, c'2=c2+k2, c'3=c3+k3]
| | cumulative([S1,S2,S3], [d1,d2,d3], [c'1,c'2,c'3], 5)
| minimize(Send)
maximize(Send)

Étant donné qu'il n'y a que des variables existentielles, la stratégie est réduite à un tronc donnant l'attaque la plus puissante et la réponse de l'ordonnanceur.

Tarification de liens. Voici un exemple venant de l'industrie des télécoms, repris de [15].

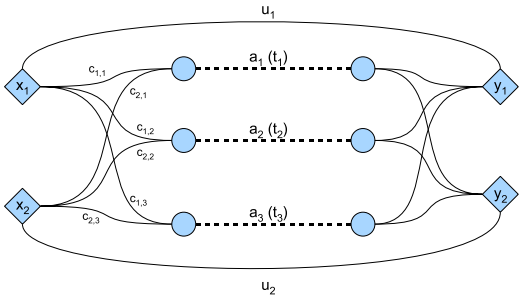

FIG. 6 – Un problème de tarification de liens

Le problème consiste à définir un tarif d'utilisation de plusieurs liens de manière à maximiser le bénéfice de leur propriétaire (le meneur). Le réseau est décrit en figure 6. il se compose de $NCustomer$ clients (les suiveurs)qui cherchent à acheminer leurs données à moindre coût, indépendamment les uns des autres. Chaque chemin d'une source vers une destination doit passer par un arc a_j, sachant que le chemin de s_i à a_j est taxé au client à hauteur de c_{ij} par un autre fournisseur d'accès. On considère que chaque client i cherche à minimiser ses propres dépenses, et qu'il peut toujours choisir une offre concurrente qui lui coûtera u_i. Le but du problème est de déterminer les tarifs t_j à appliquer de manière à maximiser le bénéfice de l'opérateur téléphonique. La figure 6 montre cette situation avec 2 clients et 3 liens. Ce problème peut s'exprimer en QCOP$^+$ comme suit :

const NCustomer
const NArc
// c[i,j] = coût pour aller de Ci à Aj
const c[NCustomer,NArc]
// d[i] = quantités de données du client i
const d[NCustomer]
// u[i] = prix maximum pour le client i
const u[NCustomer]
∃ t[1], ..., t[NArc] ∈ [0,max]
| ∀ k ∈ [1,NCustomer]

```
| | ∃ a ∈ [1,NArc],
| | | cost ∈ [1,max],
| | | income ∈ [0,max]
| | | cost = (c[k,a]+t[a])*d[k]
| | | income = t[a]*d[k]
| | | cost =< u[k]
| | minimize(cost)
| s :sum(income)
maximize(s)
```

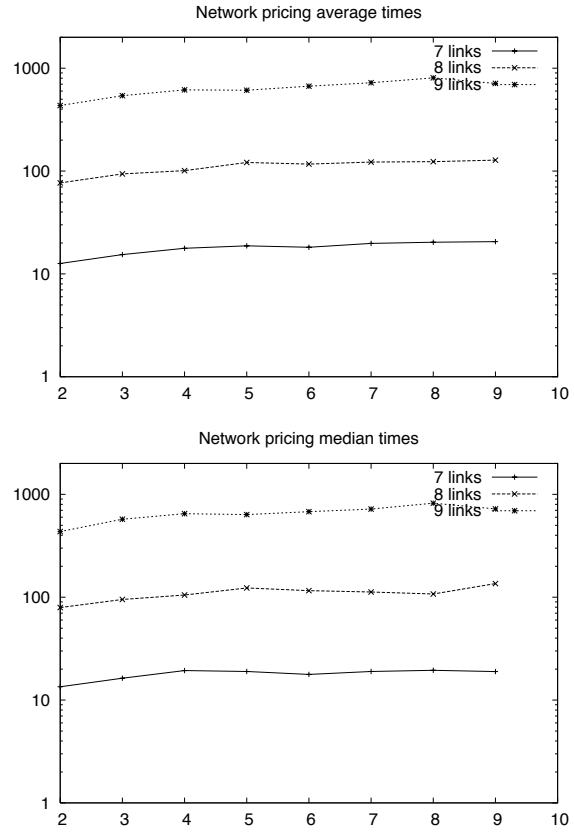

FIG. 7 – moyenne et médiane des temps de résolution (ordonnée, en secondes) du *problème de tarification de liens* pour 7, 8 et 9 liens, et pour entre 2 et 9 clients(en abscisse).

Nous avons généré aléatoirement des ensembles d'instances de ce problème. Ces ensembles diffèrent les uns des autres suivant deux paramètres : d'une part, le nombre de liens détenus par l'opérateur et d'autre part le nombre de clients voulant utiliser ces liens. L'opérateur peut choisir entre 5 tarifs possibles pour chacun de ses liens. Dans un ensemble donné, chaque instance varie sur le prix maximum acceptable par chaque client, ainsi que sur les coûts initiaux pour acheminer les données du point de départ vers le point d'entrée de chaque lien, ces données étant tirées aléatoirement. Chaque ensemble contient 100 instances.

Ces tests ont été menés sur des machines équipées chacune de deux Opteron dual-core et de 4 Go de RAM. QeCode étant mono-threadé, chacun des coeurs d'exécution traitait une instance différente. La résolution de toutes les instances a été menée à terme. La figure 7 montre les temps de résolution moyens et médians de ces tests. Les instances ayant un nombre de liens inférieur à 7 ne sont pas montrées car la majorité d'entre elles sont résolues en moins d'une seconde. On remarque que le nombre de clients influe peu sur le temps de résolution par rapport au nombre de liens. Ceci est naturel, étant donné que l'ajout d'un client conduit simplement à déterminer quel chemin il empruntera, alors que l'ajout d'un lien offre un choix de plus à chaque client et, surtout, multiplie les différents choix possibles de tarification de l'opérateur.

Tarification de réseaux virtuels. Les infrastructures des réseaux télécoms sont extrêmement coûteuses à mettre en place. de ce fait, seules une poignée d'*opérateurs télécom* (NO) les détiennent. De manière à créer de la concurrence, les gouvernement ont soutenu la mise en place d'*opérateurs* (VNO) qui fournissent les mêmes services bien qu'ils ne possèdent pas leur propre infrastructure, et louent de la bande passante au réseau d'un NO à la place. Dans un tel environnement économique, les décisions nécessitent un modèle d'oligopole complexe et loin des règles de l'équilibre général et de la concurrence parfaite de Walras. Les acteurs sont en concurrence tout en coopérant, dans des limites fixées par l'autorité de régulation. Ces exemple est tiré de [16].

La figure 8 représente les relations entre le NO, le VNO et les clients, chacun des acteurs étant modélisé dans [16]. On se place du point de vie du NO et l'objectif principal du modèle est de déterminer les décisions $y = (y_1, y_2)$, y_1 étant la fourniture de service à ses propres clients et y_2 le prix de location au VNO. Les décisions prises par le VNO sont $z = (z_1, z_2)$, z_1 étant le tarif du service à ses clients et z_2 la capacité louée au NO. Les clients sont modélisés par $n = (n_1, n_2)$ ((nombre total de clients , respectivement du NO et du VNO) en fonction des tarifs

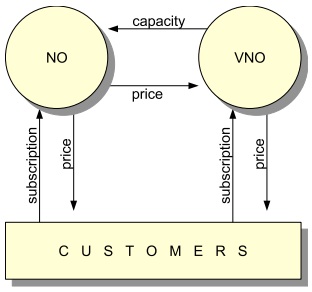

FIG. 8 – Tarification de réseaux virtuels

fixés : $n_i = k_i + r_{i,1}y_1 + r_{i,2}y_2$, les paramètres $k_i, r_{i,1}$ and $r_{i,2}$ étant déterminés par l'analyse du marché. Le bénéfice du VNO est égal au revenu tiré de ses clients moins le tarif de location au NO, c'est à dire $(q - e_2z_1)n_2 - y_2z_2 - g_2$, où q, e_2 et g_2 sont respectivement le coût variable par client et le coût fixe de fourniture de service. Notons que le revenu du NO dépend des décisions proses par le VNO. De plus, le tarif appliqué au client pour c service est compris entre des limites d et D. Le prix de location a une borne supérieure U_1 fixée par l'autorité de régulation et la capacité maximale disponible pour le VNOest inférieurs à une limite U_2 donnée.

Cela nous amène au au modèle suivant. Le quantificateur universel n'est pas utilisé, étant donné qu'il n'y a qu'un seul opérateur virtuel. Là encore, les conditions d'optimisation se chevauchent et il est donc incorrect d'utiliser le branch-and-bound.

```
const d, D, U1, U2, k1, r11, r12, r21, r22, g1, q
∃ y1 ∈ Dy1, y2 ∈ Dy2
| [d =< y1, y1 =< D, y2 =< U1]
| ∃ z1 ∈ Dz1, z2 ∈ Dz2
| | [d =< z1, z1 =< D, z2 =< U2]
| | ∃ n1 ∈ Dn1, n2 i∈ Dn2, rno ∈ Drno, rvno ∈ Drvno
| | | n1 = k1 - r11 * y1 + r12 * z1
| | | n2 = k2 + r21 * y1 - r22 * z1
| | | rvno = (q - e2 * z1) * n2 - y2 * z2 - g2
| | | rno = g1 + (q + y1 + e1) * n1 + y2 * z2
| maximize(rvno)
maximize(rno)
```

6 Discussion

On peut se demander quelle est la classe de complexité des QCOP$^+$. Cette question mériterai plus de travail théorique pour décrire l'analogue de la classe NPO pour les problèmes PSPACE.

En plus de constituer des conditions d'optimisation, min et max sont aussi des fonctions d'agrégat possibles. Il y a cependant une différence entre l'agrégat min (que l'on appelle min-agg) et la condition d'optimisation (que l'on appelle min-opt dans le sens où min-agg force toutes les branches de l'universelle à posséder des sous-stratégies alors que min-opt renvoie le minimum des branches non contredites. De plus, si le quantificateur restreint lui-même n'a pas de solution, min-agg renvoie $0_{\min}$ alors que min-opt échoue. Dans les exemples, on utilise le quantificateur existentiel autant pour le principal décideur que pour son adversaire car, le problème ne permettant pas à l'adversaire de faire complètement échouer le projet, il suffit de rechercher la stratégie optimale pour l'un comme pour l'autre.

Ce papier s'approche de plusieurs travaux : le formalisme *Plausibility-Feasibility-Utility* (PFU) [17], les *expressions itérées* [18] et les CSP stochastiques [14]. Le but du premier est de fournir une unification algébrique de plusieurs formalismes – les QBF, les QCSP les CSP stochastiques, les réseaux Bayésiens et les les processus de décision de Markov –, tandis que le deuxième cherche à modéliser l'allocation de ressource dans les workflows. Tous deux introduisent des expressions de la forme $\bigoplus_{x_1 \in D_1} \dots \bigoplus_{x_n \in D_n} expr(x_1, \dots x_n)$ avec $\bigoplus \in \{\min, \max, \sum, \Pi\}$. Il est impossible d'exprimer les modèles bi-niveaux dans ces formalismes car la condition d'optimisation doit s'appliquer sur le résultat d'une sous-expression immédiate. Cependant, des constructions comme $\min(\sum_x e_1(x) + \sum_y e_2(y))$ peuvent être exprimées par une PFU ou une expression itérée et pas par un QCOP$^+$. de plus, la condition pour que le branch-and-cound puisse s'appliquer est toujours vérifiée par construction.

[19] propose de trouver une stratégie d'un QCSP booléen qui maximise une somme pondérée ds variables existentielles ayant la valeur *vrai*. Il prouve un théorème de dichotomie permettant de classer approximativement de tels problèmes en classes praticables et impraticables. Tous ces travaux, y compris le langage des QCOP$^+$, posent le problème de trouver un langage adéquat pour exprimer le choix d'une stratégie dans les problèmes quantifiés. Quel est l'équivalent des CSP pondérés dans ce contexte ?

Quand plusieurs stratégies sont optimales au premier niveau, il peut arriver qu'elles diffèrent considérablement au niveau des sous-stratégies induites. Le langage des QCOP$^+$ ne permet pas pour le moment d'exprimer ce genre de "meta-optimisation". Un autre problème serait de permettre de poster des contraintes entre des variables et des résultats d'agrégat. Il est par exemple impossible d'exprimer le fait que les sous-stratégies doivent attribuer une valeur différente à une variable existentielle pour chaque valeur d'une variable universelle : cela nécessiterait une contrainte agrégat "Alldifferent" qui pourrait échouer. L'évaluation des stratégies partielles et/ou non-gagnantes reste aussi une question ouverte qui pourrait ouvrir la voie à l'adaptation de la relaxation de contraintes et à la recherche locale dans les problèmes quantifiés.

7 Conclusion

Nous avons présenté le formalisme des QCOP$^+$ pour modéliser des problèmes d'optimisation quantifiés. Ce cadre est suffisamment large pour englober les problèmes bi-niveaux (et multi-niveaux) qui sont étudiés en théorie des jeux et en recherche opérationnelle depuis des années. Les quantificateurs existentiels

peuvent être dotés d'une condition d'optimisation qui permet de choisir une stratégie optimisant un paramètre, et les quantificateurs universels sont utilisés pour calculer des agrégats. Ce travail fournit un moyen d'exprimer et de résoudre de tels problèmes et étends les QCSP à des problèmes plus généraux que l'étude du pire cas.

Ce travail est supporté par le projet ANR-06-BLAN-0383.

Références

[1] Nightingale, P. : Consistency for quantified constraint satisfaction problems. In van Beek, P., ed. : CP. Volume 3709 of Lecture Notes in Computer Science., Springer (2005) 792–796

[2] Bessière, C., Verger, G. : Strategic constraint satisfaction problems. In Miguel, I., Prestwich, S., eds. : Workshop on Constraint Modelling and Reformulation, Nantes, France (2006) 17–29

[3] Benedetti, M., Lallouet, A., Vautard, J. : QCSP Made Practical by Virtue of Restricted Quantification. In Veloso, M., ed. : International Joint Conference on Artificial Intelligence, Hyderabad, India, AAAI Press (2007) 38–43

[4] Benedetti, M., Lallouet, A., Vautard, J. : Modeling adversary scheduling with QCSP+. In : ACM Symposium on Applied Computing, Fortaleza, Brazil, ACM Press (2008)

[5] Nightingale, P. : Consistency and the Quantified Constraint Satisfaction Problem. PhD thesis, University of St Andrews (2007)

[6] Bracken, J., McGill, J. : Mathematical programs with optimization problems in the constraints. Operations Research **21** (1973) 37–44

[7] Stackelberg, H. : The theory of market economy. Oxford University Press (1952)

[8] Bialas, W.F. : Multilevel mathematical programming, an introduction. Slides (2002)

[9] Colson, B., Marcotte, P., Savard, G. : An overview of bilevel optimization. Annals of Operations Research **153** (2007) 235–256

[10] QeCode Team : QeCode : An open QCSP+ solver (2008) Available from `http ://www.univ-orleans.fr/lifo/software/qecode/`.

[11] Gecode Team : Gecode : Generic constraint development environment (2006) Available from `http ://www.gecode.org`.

[12] Benedetti, M. : Extracting certificates from quantified boolean formulas. In Kaelbling, L.P., Saffiotti, A., eds. : International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, Professional Book Center (2005) 47–53

[13] Bordeaux, L., Cadoli, M., Mancini, T. : CSP properties for quantified constraints : Definitions and complexity. In Veloso, M.M., Kambhampati, S., eds. : National Conference on Artificial Intelligence, AAAI Press (2005) 360–365

[14] Walsh, T. : Stochastic constraint programming. In : ECAI. (2002) 111–115

[15] Bouhtou, M., Grigoriev, A., van Hoesel, S., van der Kraaij, A.F., Spieksma, F.C., Uetz, M. : Pricing bridges to cross a river. Naval Research Logistics **54**(4) (2007) 411–420

[16] Audestad, J.A., Gaivoronski, A.A., Werner, A. : Extending the stochastic programming framework for the modeling of several decision makers : pricing and competition in the telecommunication sector. Annals of Operations Research **142**(1) (2006) 19–39

[17] Pralet, C., Verfaillie, G., Schiex, T. : An algebraic graphical model for decision with uncertainties, feasibilities, and utilities. Journal of Artificial Intelligence Research **29** (2007) 421–489

[18] Bordeaux, L., Hamadi, Y., Quimper, C.G., Samulowitz, H. : Expressions Itérées en Programmation par Contraintes. In Fages, F., ed. : Journées Francophones de Programmationpar Contraintes. (2007) 98–107

[19] Chen, H., Pál, M. : Optimization, games, and quantified constraint satisfaction. In Fiala, J., Koubek, V., Kratochvíl, J., eds. : MFCS. Volume 3153 of Lecture Notes in Computer Science., Springer (2004) 239–250