Compilation des QCSP

Igor Stéphan

Compilation des QCSP

Igor Stéphan

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045, Angers, Cedex 01, France
igor.stephan@info.univ-angers.fr

Résumé

Nous proposons dans cet article un cadre formel pour la compilation des problèmes de satisfaction de contraintes quantifiées (QCSP). L'objectif d'une telle compilation est de répondre au problème du choix du prochain mouvement de manière polynomiale en temps même lorsque la solution courante n'est plus accessible. Nous établissons la sémantique de ce formalisme en terme d'interprétation en un QCSP. Nous en étudions les propriétés en particulier vis-à-vis du QCSP compilé. Nous spécifions deux algorithmes de compilation basés sur un algorithme de recherche. Le premier est imbriqué dans l'algorithme de recherche et reprend la structure inductive de la sémantique des QCSP ; le second est un analyseur de trace d'exécution d'un solveur QCSP.

Abstract

We propose in this article a framework for compilation of quantified constraint satisfaction problems (QCSP). The aim of this compilation is to answer the next move choice problem in polytime even when the current solution is no more accessible. We establish the semantics of this formalism by an interpretation to a QCSP. We specify two algorithms based on search algorithm. The first one is embedded into the search algorithm and is based on the inductive semantics of the QCSP. The second one is a QCSP solver trace analyser.

1 Introduction

Un problème de satisfaction de contraintes (ou CSP pour *Constraint Satisfaction Problem*) [13] est le problème de trouver une affectation de valeurs pour des variables qui satisfont un ensemble de contraintes. Un problème de satisfaction de contraintes *quantifiés* (ou QCSP pour *Quantified Constraint Satisfaction Problem*) [7] est une extension du problème de satisfaction de contraintes dans laquelle certaines variables sont quantifiées universellement (les autres variables étant quantifiées existentiellement) ; dans ce cadre, chaque variable prend sa valeur dans un domaine fini discret. Les variables universellement quantifiées peuvent être considérées comme étant une forme d'incertitude : les caprices de la nature ou les choix d'un adversaire dans un jeu à deux joueurs. Dans la seconde interprétation, calculer une solution à un QCSP c'est offrir au joueur existentiel (i.e. le joueur dont les coups sont représentés par des variables existentiellement quantifiées) une stratégie qui gagne à tous les coups. Tandis que résoudre un CSP est en général NP-complet, résoudre un QCSP est PSPACE-complet. La plupart des procédures de décision récentes pour les QCSP [1, 4, 11, 5] sont basées sur un algorithme de recherche [1]. Un tel algorithme choisit une variable, teste en affectant à la variable les différentes valeurs du domaine si les QCSP admette une solution et combine les résultats selon la sémantique du quantificateur associé à la variable.

En général, une base de connaissances est compilée une bonne fois pour toute dans un langage cible puis utilisée à la volée pour répondre à des questions ; l'objectif étant d'avoir une complexité bien moindre pour l'interrogation de la base de connaissances compilée que celle non compilée. L'objectif de notre compilation est de répondre au problème du choix du prochain mouvement de manière polynomiale en temps même lorsque la solution courante n'est plus accessible. Notre but est donc bien de compiler le QCSP et non simplement une solution. De part la complexité du calcul d'une simple solution, la recherche d'une forme compilée d'un QCSP est capitale. À notre connaissance, ce problème n'a pas été abordé dans le cadre des QCSP mais seulement dans le cadre de domaines connexes : la compilation des formules booléennes quantifiées (ou QBF pour *Quantified Boolean Formulas*) [16, 15, 10], la compilation des bases de connaissances [6, 9].

Ce présent article est une reformulation dans ces principes de [16, 15] mais est techniquement complè-

1. hormis [17] qui est basé sur une approche *bottom-up* et [12] qui est basé sur une traduction vers les formules booléennes quantifiées.

tement différent et novateur. Cet article est organisé ainsi : la section 2 pose l'ensemble des définitions nécessaires ; la section 3 définit notre proposition de cadre formel pour la compilation des QCSP ; la section 4 spécifie deux manières différentes et complémentaires de construire des bases pour un QCSP ; la section 5 dresse une conclusion et quelques perspectives.

2 Préliminaires

Le symbole $\exists$ représente le quantificateur existentiel et le symbole $\forall$ représente le quantificateur universel. Le symbole $\wedge$ représente la conjonction, le symbole $\top$ représente ce qui est toujours vrai et le symbole $\bot$ représente ce qui est toujours faux. Un QCSP est un n-uplet $(\mathbf{V}, \text{ordre}, \text{quant}, \mathbf{D}, \mathcal{C})$ dans lequel $\mathbf{V}$ est un ensemble de n variables, ordre est une bijection des variables sur $[1..n]$, quant est fonction de $\mathbf{V}$ dans $\{\exists, \forall\}$ (quant(v) est le quantificateur associé à la variable v), $\mathbf{D}$ est une fonction de l'ensemble de variables dans un ensemble de domaines $\{D(v_1), \ldots, D(v_n)\}$ où, pour chaque variable $v_i \in \mathbf{V}$, $D(v_i)$ est le domaine fini de l'ensemble des valeurs possibles ($D(v)$ est le domaine associé à la variable v), $\mathcal{C}$ est un ensemble de contraintes. Si $v_{j_1}, \ldots, v_{j_m}$ sont des variables d'une contrainte $c_j \in \mathcal{C}$ alors la relation associée à c_j est un sous-ensemble du produit cartésien $D(v_{j_1}) \times \ldots \times D(v_{j_m})$. Par la suite, nous noterons pour tout $i \in [1..n]$, $q_i = \text{quant}(v_i)$ et $D_i = D(v_i)$. Un QCSP $(\mathbf{V}, \text{ordre}, \text{quant}, \mathbf{D}, \mathcal{C})$ sur n variables sera décrit de façon plus compacte ainsi :

$$q_1 v_1 \ldots q_n v_n \bigwedge_{c_j \in \mathcal{C}} c_j$$

avec $v_1 \in D_1$, …, $v_n \in D_n$, ordre$(v_i) = i$, pour tout $i \in [1..n]$; $q_1 v_1 \ldots q_n v_n$ forme le lieur ; un lieur vide est noté ε.

Le QCSP $(\{x, y, z, t\}, \text{ordre}, \text{quant}, \mathbf{D}, \mathcal{C})$ avec

$$\begin{cases} \text{ordre} = \{(1, x), (2, y), (3, z), (4, t)\}, \\ \text{quant} = \{(x, \exists), (y, \exists), (z, \forall), (t, \exists)\}, \\ \mathbf{D} = \{(x, D(x)), (y, D(y)), (z, D(z)), (t, D(t))\}, \\ D(x) = D(y) = D(z) = D(t) = \{0, 1, 2\}, \\ \mathcal{C} = \{(x = (y * z) + t)\} \end{cases}$$

est, par exemple, noté : $\exists x \exists y \forall z \exists t (x = (y * z) + t)$ avec $x, y, z, t \in \{0, 1, 2\}$.

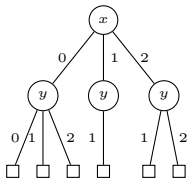
Dans la partie algorithmique (cf section 4), une variable v associée à un domaine D est notée v_D.

Dans un lieur, une séquence maximale homogène de quantificateurs forme un bloc ; le premier (et aussi le plus externe) est le plus à gauche.

L'ensemble $\mathcal{A}_i(Q)$ avec $v_1 \in D_1$, …, $v_n \in D_n$, $Q = q_1 v_1 \ldots q_n v_n$ pour $1 \leq i \leq n$ est l'ensemble des arbres dont

- toutes les feuilles sont étiquetées par le symbole $\square$ et à la profondeur i,
- tout nœud interne de profondeur k, $0 \leq k < i$ est étiqueté par la variable v_{k+1},
- tout arc reliant un nœud de profondeur k à un de ses nœuds fils est étiqueté par un élément de D_{k+1},
- tous les arcs reliant un nœud de profondeur k aux nœuds fils sont étiquetés différemment.

L'arbre suivant est, par exemple, un élément de l'ensemble $\mathcal{A}_2(\exists x \exists y \forall z \exists t)$ avec $x, y, z, t \in \{0, 1, 2\}$:



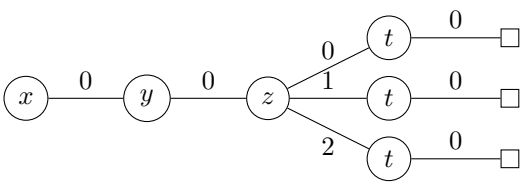
Soit un QCSP $(\mathbf{V}, \text{ordre}, \text{quant}, \mathbf{D}, \mathcal{C})$ tel que $\mathbf{V} = \{v_1, \ldots, v_n\}$, avec $v_1 \in D_1$, …, $v_n \in D_n$, alors un scénario est la séquence des étiquettes $val_1, \ldots, val_n$ sur une branche $(v_1, val_1), \ldots, (v_n, val_n)$, $val_i \in D_i$ pour tout i, $1 \leq i \leq n$, d'un arbre de $\mathcal{A}_n(q_1 v_1 \ldots q_n v_n)$ et une stratégie est un arbre de $\mathcal{A}_n(q_1 v_1 \ldots q_n v_n)$ tel que

- pour tous les nœuds étiquetés par une variable dont le quantificateur associé est existentiel admettent un unique nœud fils et
- pour tous les nœuds étiquetés par une variable dont le quantificateur associé est universel et le domaine associé est de taille k, admettent k nœuds fils.

Un scénario $val_1, \ldots, val_n$ pour un QCSP $(\mathbf{V}, \text{ordre}, \text{quant}, \mathbf{D}, \mathcal{C})$ tel que $\mathbf{V} = \{v_1, \ldots, v_n\}$ est un scénario gagnant si $(\bigwedge_{1 \leq i \leq n} v_i = val_i) \wedge (\bigwedge_{c_j \in \mathcal{C}} c_j)$ est vrai ; un tel scénario correspond à l'instantiation complète $[v_1 \leftarrow val_1]$, …, $[v_n \leftarrow val_n]$ qui est gagnante si l'instantiation satisfait toutes les contraintes. Une stratégie est gagnante si tous ses scénarios sont gagnants. En l'absence de quantificateur, la stratégie $\square$ est toujours une stratégie gagnante.

La scénario $0, 0, 2, 0$, qui correspond à l'instantiation complète $[x \leftarrow 0]$, $[y \leftarrow 0]$, $[z \leftarrow 2]$ et $[t \leftarrow 0]$, est un scénario gagnant, car $0 = (0 * 2) + 0$, ainsi que la stratégie suivante pour le QCSP

$$\exists x \exists y \forall z \exists t (x = (y * z) + t), x, y, z, t \in \{0, 1, 2\}$$



car $0 = (0 * 0) + 0$, $0 = (0 * 1) + 0$ et $0 = (0 * 2) + 0$.

Un QCSP $\forall x QC$ avec $x \in D$ admet une stratégie gagnante si et seulement si, pour tout $val \in D$, $Q(C \wedge (x = val))$ admet une stratégie gagnante et un QCSP $\exists x QC$ avec $x \in D$ admet une stratégie gagnante si et seulement si, pour au moins un $val \in D$, $Q(C \wedge (x = val))$ admet une stratégie gagnante.

3 Base pour QCSP

Une manière naïve de compiler un QCSP serait de stocker l'ensemble des stratégies gagnantes dans un ensemble mais l'approche est, du point de vue de la complexité spatiale, fortement exponentielle [2] et donc inappropriée ; par exemple, pour le QCSP $\forall x \forall y \exists z \exists t (x = (y * z) + t), x, y, z, t \in \{0, 1, 2\}$ il existe 324 stratégies gagnantes [3] ! Une autre manière de faire est de stocker un arbre contenant uniquement les scénarios présents dans les stratégies gagnantes ; cette approche est pauvre du point de vue de la représentation des connaissances puisqu'il n'y a pas d'accès direct aux possibilités d'une variable existentiellement quantifiée, hormis celles du premier bloc.

Nous définissons dans cette section notre formalisme pour représenter le résultat de la compilation d'un QCSP : la « base » ainsi que sa sémantique en terme de QCSP.

3.1 Définitions

Une base est intuitivement l'ensemble des stratégies organisées selon un mécanisme de garde pour chaque variable existentiellement quantifiée et chaque valeur du domaine de cette variable ; une telle garde est un arbre qui est l'expression des coups joués par les deux adversaires.

Définition 1 (base) *Une base est soit*
- *le symbole top_base*
- *le symbole bottom_base*
- *une paire $\langle Q \mid G \rangle$ avec $n > 0$, $Q = q_1 v_1 \dots q_n v_n$ et $G = [G_{e_1}, \dots, G_{e_m}]$ une liste telle que*
  - *$e_1, \dots, e_m$ est l'ensemble des indices des variables quantifiées existentiellement [4] ;*
  - *chaque G_{e_k}, $1 \leq k \leq m$ est une fonction dont le graphe $\{(val_1, a_1), \dots, (val_{j_k}, a_{j_k})\}$ est non vide, $val_1, \dots, val_{j_k} \in D(v_{e_k})$ et $a_1, \dots, a_{j_k} \in \mathcal{A}_{e_k}(Q)$.*

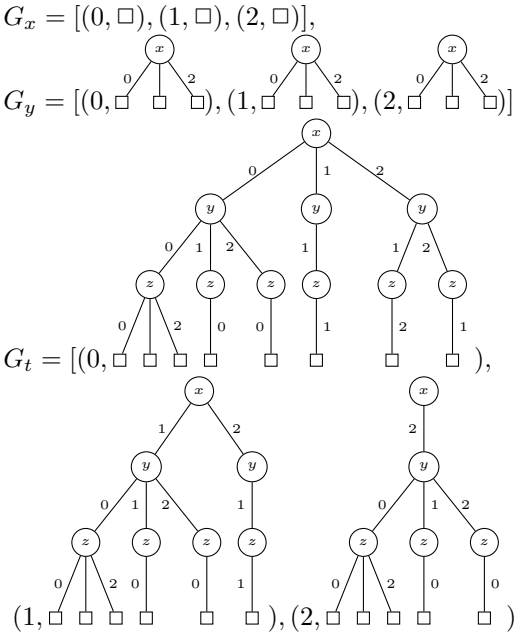


FIGURE 1 – Ensembles de gardes.

Dans ce qui suit, les bases *top_base* et *bottom_base* sont interprétées sémantiquement comme respectivement ce qui est toujours vrai et ce qui est toujours faux et algorithmiquement comme respectivement ce qui admet tout pour stratégie gagnante et ce qui n'admet aucune stratégie gagnante.

Exemple 1 *La figure 1 présente les ensembles de gardes G_x, G_y et G_z de la base $B = \langle \exists x \exists y \forall z \exists t \mid [G_x, G_y, G_t] \rangle$ avec $x, y, z, t \in \{0, 1, 2\}$.*

Nous avons choisi d'explorer uniquement les gardes sous la forme d'arbre mais il fait tout à fait possible, au prix d'une algorithmique plus complexe et d'un propos qui ne correspondait pas à ce qui voulait être présenté dans cet article, de les remplacer par, par exemple, des ADD (pour *Algebraic Decision Diagram* [2]) pour compiler aussi les gardes.

3.2 Interprétation

La sémantique d'une base s'exprime via une interprétation vers les QCSP. Nous interprétons tout d'abord les arbres présents dans les bases comme des contraintes sous formes de tables (i.e. des n-uplets de valeurs).

2. Dans les hypothèses de complexité classiques, la taille d'une quelconque représentation d'une stratégie gagnante est dans le pire des cas exponentielle par rapport au nombre de variables du QCSP [8]

3. Si n est la taille du domaine des variables, la fonction $\#(.)$ qui associe à un lieur Q son nombre de stratégies possibles est définie par $\#(\varepsilon) = 1$, $\#(\exists x Q) = n \times \#(Q)$ et $\#(\forall x Q) = \#(Q)^n$; donc pour le lieur $\forall x \forall y \exists z \exists t$ et des domaines pour les variables x, y, z et t de taille 3, il y $\#(\forall x \forall y \exists z \exists t) = ((3 \times 3 \times 1)^3)^3 = 387420489$ stratégies possibles.

4. i.e. $u_1, \dots, u_p$ est l'ensemble des indices des variables quantifiées universellement, $\{e_1, \dots, e_m\} \cup \{u_1, \dots, u_p\} = [1..n]$, $\{e_1, \dots, e_m\} \cap \{u_1, \dots, u_p\} = \emptyset$, $\text{quant}(v_{e_i}) = \exists$, pour tout i, $1 \leq i \leq n$, $\text{quant}(v_{u_i}) = \forall$, pour tout i, $1 \leq i \leq p$

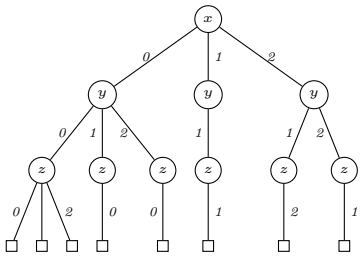
Définition 2 (interprétation d'un arbre) *L'interprétation d'un arbre a selon une valeur val est l'ensemble*

$I^{val}(a) = \{(val, e_1, \ldots, e_n) | e_1 \ldots e_n$
un chemin de la racine à une feuille de l'arbre a$\}$.

En particulier, $I^{val}(\Box) = \{val\}$. *L'interprétation d'un ensemble G de couples formés par une valeur et un arbre est par extension :*

$$I(G) = \bigcup_{(val,a)\in G} I^{val}(a).$$

Exemple 2 *En continuant l'exemple 1, l'interprétation de l'arbre a extrait de* G_t *:*



selon la valeur 0 est l'ensemble

$I^0(a) = \{(0,0,0,0), (0,0,0,1), (0,0,0,2), (0,0,1,0),$
$(0,0,2,0), (0,1,1,1), (0,2,1,2), (0,2,2,1)\}$.

et

$I(G_t) = \{(0,0,0,0), (0,0,0,1), (0,0,0,2),$
$(0,0,1,0), (0,0,2,0), (0,1,1,1), (0,2,1,2),$
$(0,2,2,1), (1,1,0,0), (1,1,0,1), (1,1,0,2),$
$(1,1,1,0), (1,1,2,0), (1,2,1,1), (2,2,0,0),$
$(2,2,0,1), (2,2,0,2), (2,2,1,0), (2,2,2,0)\}$

Nous sommes alors en mesure de définir l'interprétation d'une base.

Définition 3 (interprétation d'une base) *La fonction* $(\cdot)^*$ *d'interprétation d'une base en un QCSP est définie par* $(Q = q_1 v_1 \ldots q_n v_n)$ *:*

$(top\_base)^* = \top$
$(bottom\_base)^* = \bot$
$(\langle Q \mid [G_{e_1}, \ldots, G_{e_m}]\rangle)^* =$
$Q \bigwedge_{e_k \in [e_1,\ldots,e_m]} ((v_{e_k}, v_1, \ldots, v_{e_k - 1}) \in I(G_{e_k}))$

Exemple 3 *En continuant les exemples 1 et 2*

$(B)^* =$
$\exists x \exists y \forall z \exists t$
$((x \in I(G_x)) \wedge ((y,x) \in I(G_y)) \wedge ((t,x,y,z) \in I(G_t)))$

avec $x, y, z, t \in \{0,1,2\}$, $I(G_x) = \{0,1,2\}$ *et* $I(G_y) = \{0,1,2\}^2$.

3.3 Propriétés

À un QCSP donné, plusieurs bases peuvent correspondre dans le sens où l'interprétation de celles-ci a exactement les mêmes stratégies gagnantes que le QCSP, c'est ce que nous définissons comme la compatibilité.

Définition 4 (compatibilité d'une base) *Une base est compatible avec un QCSP si l'interprétation de celle-ci a exactement les mêmes stratégies gagnantes.*

Exemple 4 *En continuant les exemples 1, 2 et 3, la base B est compatible avec le QCSP* $\exists x \exists y \forall z \exists t (x = (y * z) + t)$ *avec* $x, y, z, t \in \{0,1,2\}$. *Si, par exemple, le couple* $(2, \Box)$ *de* G_x *est oté alors la base résultante n'est plus compatible car il lui manque deux des quatre stratégies gagnantes.*

Le théorème suivant est un théorème immédiat de complétude.

Théoreme 1 (complétude) *Pour tout QCSP il existe une base qui lui est compatible.*

Dans le cas d'un QCSP représentant un jeu à deux joueurs, une des questions les plus importantes, à chaque tour de jeu, pour le joueur existentiel est la suivante : « Que dois-je jouer pour être certain de gagner la partie ? ». Si une stratégie gagnante a été précalculée alors le joueur n'a qu'à suivre les prescriptions de celle-ci mais dans le cas où l'incertitude n'a pas été entièrement prise en compte et que la stratégie gagnante courante n'est plus d'actualité, le joueur doit recalculer entièrement une nouvelle stratégie gagnante et en payer le prix.

Définition 5 (prochain mouvement) *Instance : Un QCSP* $q_1 v_1 \ldots q_n v_n \bigwedge_{c\in\mathcal{C}} c$ *avec* $v_1 \in D_1$, *...,* $v_n \in D_n$ *et une séquence de branchements* $v_1 = val_1$, *...,* $v_i = val_i$ *obtenue d'une solution au QCSP avec quant*$(v_1) = \exists$ *et* $val_1 \in D_1$, *...,* $val_i \in D_i$.
Question : Existe-t-il une stratégie gagnante pour un QCSP

$q_{i+1} v_{i+1} \ldots q_n v_n \bigwedge_{c\in\mathcal{C}} c \wedge$
$(v_1 = val_1) \wedge \ldots \wedge (v_{i-1} = val_{i-1}) \wedge (v_i = val'_i)$

avec $v_{i+1} \in D_{i+1}$, *...,* $v_n \in D_n$, $val'_i \in D_i$, $val'_i \neq val_i$.

Clairement, dans le cas général, le problème du choix du prochain mouvement est encore un problème PSPACE-complet.

Nous introduisons la propriété d'optimalité pour une base qui va garantir que le problème du choix du prochain mouvement n'est plus un problème PSPACE-complet mais polynomial en temps par rapport à la taille de la base. Une base n'est dite *optimale* que si toute garde, qui est représentée sous forme d'arbre, associée à une valeur pour une variable, garantit que si la garde est passante pour les coups déjà joués (i.e. ils forment une branche de l'arbre ou encore, ce qui est équivalent, ils forment un n-uplet appartenant à l'interprétation de l'arbre selon la valeur) alors en jouant ce coup pour cette variable le joueur existentiel est sûr de continuer dans une stratégie gagnante.

Définition 6 (optimalité) *Soit une base* $B = \langle q_1 v_1 \dots q_n v_n \mid [G_{e_1}, \dots, G_{e_m}] \rangle$ *et* $(B)^* = q_1 v_1 \dots q_n v_n C_G$ *avec* $v_1 \in D_1$, *...*, $v_n \in D_n$. *La base est optimale si la propriété suivante est vérifiée. Pour tout* i, $i \in [1 \dots m]$, *soit* C_i *l'ensemble de contraintes* $\{(v_{e_k} = val_{e_k}) | 1 \leq k < i\}$ *telles que* $(val_{e_k}, val_{e_1}, \dots, val_{e_{k-1}}) \in I^{val_{e_k}}(a_{e_k})$, $(val_{e_k}, a_{e_k}) \in G_{e_k}$. *Alors pour tout couple* $(val, a) \in G_{e_i}$, $(val, val_{e_1}, \dots, val_{e_{i-1}}) \in I^{val}(a)$ *si et seulement si* $q_{e_i+1} v_{e_i+1} \dots q_n v_n (C_G \wedge (v^{e_i} = val) \wedge \bigwedge_{c \in C_i} c)$ *admet une stratégie gagnante.*

Exemple 5 *La figure 2 présente les ensembles de gardes* G_x^{opt}, G_y^{opt} *et* G_t^{opt} *de la base* $B^{opt} = \langle \exists x \exists y \forall z \exists t \mid [G_x^{opt}, G_y^{opt}, G_t^{opt}] \rangle$ *qui est optimale et compatible avec le QCSP :*

$$\exists x \exists y \forall z \exists t (x = (y * z) + t)$$

avec $x, y, z, t \in \{0, 1, 2\}$.

Nous explicitons ci-dessous l'optimalité de la base mais en faisant appel pour simplifier à la contrainte du QCSP qui lui est compatible.

- $i = 1$ *(i.e.* $v_{e_i} = x$*) alors* $C_i = \emptyset$ *et pour tout* $K \in \{0, 1, 2\}$, $K \in I^K(\square) = \{K\}$ *si et seulement si* $\exists y \forall z \exists t (x = (y * z) + t) \wedge (x = K)$, *avec* $y, z, t \in \{0, 1, 2\}$, *admet une stratégie gagnante.*
- $i = 2$ *(i.e.* $v_{e_i} = y$*) alors*
  - *pour tout* $K \in \{0, 1, 2\}$ $(K, \square) \in G_x^{opt}$, $I^0(T_0^y) = \{(0, 0), (0, 1), (0, 2)\}$ *et pour tout* $K \in \{0, 1, 2\}$, $(0, K) \in I^0(T_0^y)$ *si et seulement si* $\forall z \exists t (x = (y * z) + t) \wedge (y = 0) \wedge (x = K)$, *avec* $z, t \in \{0, 1, 2\}$, *admet une stratégie gagnante ;*
  - $(1, \square) \in G_x^{opt}$, $I^1(T_1^y) = \{(1, 1)\}$ *et* $(1, 1) \in I^1(T_1^y)$ *si et seulement si* $\forall z \exists t (x = (y * z) + t) \wedge (y = 1) \wedge (x = 1)$, *avec* $z, t \in \{0, 1, 2\}$, *admet une stratégie gagnante ;* $(1, 0) \notin I^1(T_1^y)$ *et* $\forall z \exists t (x = (y * z) + t) \wedge (y = 1) \wedge (x = 0)$, *avec* $z, t \in \{0, 1, 2\}$, *n'admet pas de stratégie gagnante ;* $(1, 2) \notin I^1(T_1^y)$ *et* $\forall z \exists t (x = (y * z) + t) \wedge (y = 1) \wedge (x = 2)$, *avec* $z, t \in \{0, 1, 2\}$, *n'admet pas de stratégie gagnante ;*

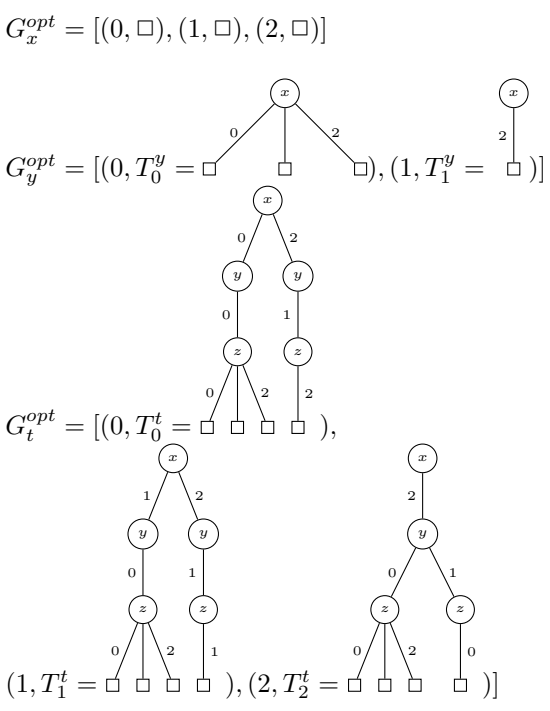


FIGURE 2 – Ensembles de gardes pour une base optimale.

  - *pour* $y = 2$, *il n'y a pas de couple* $(2, T_2^y) \in G_y$ *et* $\exists x \forall z \exists t (x = (y * z) + t) \wedge (y = 2)$, *avec* $x, z, t \in \{0, 1, 2\}$, *n'admet pas de stratégie gagnante.*
- $i = 3$ *(i.e.* $v_{e_i} = t$*) alors (nous ne traitons que le cas* $t = 0$, *les autres cas sont similaires)*
  - $(2, \square) \in G_x^{opt}$, $(1, T_1^y) \in G_y^{opt}$ *avec* $(1, 2) \in I^1(T_1^y)$ *et* $(0, 2, 1, 2) \in I^0(T_0^t)$ *si et seulement si* $(x = (y * z) + t) \wedge (x = 2) \wedge (y = 1) \wedge (z = 2) \wedge (t = 0)$ *admet une stratégie gagnante ; il en est de même avec* $(0, 0, 0, 0)$, $(0, 0, 0, 1)$ *et* $(0, 0, 0, 2)$ *;*
  - *dans tous les autres cas* $(0, val_x, val_y, val_z) \notin I^0(T_0^t)$ *et* $(x = (y * z) + t) \wedge (x = val_x) \wedge (y = val_y) \wedge (z = val_z) \wedge (t = 0)$ *n'admet pas de stratégie gagnante.*

Le théorème suivant montre tout l'intérêt de calculer une base optimale compatible avec un QCSP pour parcourir les stratégies gagnantes de ce dernier.

Théoreme 2 (prochain mouvement) *Le problème du choix du prochain mouvement pour l'interprétation d'une base optimale est polynomiale en temps par rapport à la taille de la base.*

Opérationnellement, à chaque tour de jeu, le joueur existentiel possédant une base optimale compatible avec la QCSP représentant le jeu, n'a qu'à vérifier que les coups déjà joués forment une branche de l'arbre de la garde associée au coup qu'il souhaite jouer.

4 Construction d'une base

Nous proposons deux algorithmes de construction d'une base dans le cadre d'un algorithme de recherche : le premier, récursif, est imbriqué dans ce dernier et reprend la structure inductive de la sémantique des QCSP, il est décrit dans la sous-section 4.1 ; le second, itératif, est un analyseur de trace d'exécution d'un solveur QCSP, il est décrit dans la sous-section 4.2.

4.1 Construction récursive d'une base

Algorithme 1 *rec_comp*

Entrée: Q : un lieur d'un QCSP
Entrée: C : un ensemble de contraintes d'un QCSP
Sortie: une base ou *top_base* ou *bottom_base*

si $calcul\_pointfixe\_propagation(C) = failure$ **alors**
  retourner *bottom_base*
fin si
si $vide(Q)$ **alors retourner** *top_base* **fin si**
$qx_D \leftarrow tete(Q); listeValBase \leftarrow []; d \leftarrow D$
tant que $!vide(d)$ **faire**
  $val \leftarrow tete(d); d \leftarrow queue(d)$
  $base \leftarrow rec\_comp(queue(Q), C \cup \{x = val\})$
  si $base = bottom\_base \& q = \forall$ **alors**
    retourner *bottom_base*
  fin si
  $listeValBase \leftarrow [(val, base)|listeValBase]$
fin tant que
si $vide(listeValBase)$ **alors**
  retourner *bottom_base*
fin si
si $q = \exists$ **alors**
  retourner $\oplus_\exists(x_D, queue(Q), listeValBase)$
sinon
  retourner $\oplus_\forall(x_D, queue(Q), listeValBase)$
fin si

La recherche d'une stratégie gagnante est définie de manière naturelle récursivement par la recherche simultanée sur les valeurs possibles de la variable ; l'algorithme 1 *rec_comp* de calcul d'une base compatible à partir d'un QCSP reproduit ce schéma. Il réalise en premier le calcul de point fixe de la propagation grâce à la fonction *calcul_pointfixe_propagation* sur l'ensemble des contraintes et retourne *bottom_base* si une contradiction est détectée ; si ce n'est pas le cas et que le lieur est vide alors *top_base* est renvoyé ; sinon, pour chaque valeur *val* du domaine de la variable x la plus externe du lieur, la contrainte $(x = val)$ est rajoutée aux contraintes et l'algorithme est appelé récursivement ; si la variable est associée à un quantificateur universel, il suffit qu'un des appels renvoient *bottom_base* pour que *bottom_base* soit renvoyé à son tour ; si la variable est associée à un quantificateur existentiel, il faut que tous les appels renvoient *bottom_base* pour que *bottom_base* soit renvoyé ; dans tous les autres cas, l'opérateur $\oplus_\exists$ ou $\oplus_\forall$ est invoqué pour combiner les bases entre-elles.

Algorithme 2 $\oplus_\exists$

Entrée: x_D : une variable et son domaine
Entrée: Q : un lieur
Entrée: l : une liste de paires (valeur,base)
Sortie: une base

si $constantes(l)$ **alors**
  $g \leftarrow cas\_de\_base(l)$
  si $vide(g)$ **alors retourner** *bottom_base* **sinon retourner** $\langle \exists x_D Q \mid [g] \rangle$ **fin si**
sinon
  retourner $\langle \exists x_D Q \mid [premiere(l)| \oplus (x, l)] \rangle$
fin si

Algorithme 3 $\oplus_\forall$

Entrée: x_D : une variable et son domaine
Entrée: Q : un lieur
Entrée: l : une liste de paires (valeur,base)
Sortie: une base ou *top_base*

si $constantes(l)$ **alors**
  retourner *top_base*
sinon
  retourner $\langle \forall x_D Q \mid \oplus (x, l) \rangle$
fin si

Les opérateurs $\oplus_\forall$ et $\oplus_\exists$ spécifiés respectivement par les algorithmes 2 et 3 opèrent ainsi : Pour le cas de base des deux algorithmes, la fonction *constantes* teste si la liste de couples passée en argument ne contient que des *top_base* ou des *bottom_base* pour second élément ; en cas de succès pour l'opérateur $\oplus_\forall$, il ne peut s'agir que de *top_base* dans le cas d'un bloc de quantificateurs universels le plus à l'intérieur et *top_base* est retourné ; en cas de succès pour l'opérateur $\oplus_\exists$, il ne peut s'agir que du quantificateur existentiel le plus interne et une base ne contenant que les valeurs associées aux *top_base* est construite grâce à la fonction *cas_de_base* définie par $cas\_de\_base(l) = \{(val, \square) | (val, top\_base) \in l\}$. En cas d'échec pour l'opérateur $\oplus_\exists$, la base renvoyée est construite en ajoutant au résultat de l'opérateur $\oplus$ la liste des valeurs associées aux bases grâce à la fonction *premiere* définie par $premiere(l) = \{(val, \square) | (val, a) \in l\}$. En cas d'échec pour l'opérateur $\oplus_\forall$, l'opérateur $\oplus$ spécifié par l'algorithme 4 est appliqué et le résultat est empaqueté dans une base qui est elle-même retournée puisque les

variables universelles quantifiées ne sont pas associées à des gardes.

Algorithme 4 $\oplus$

Entrée: x : une variable
Entrée: lvb : liste de paires (valeur,base)
Sortie: une liste de gardes
  $lg \leftarrow []$
  $lvg \leftarrow extrait\_gardes(lvb)$
  tant que $!vide(lvg)$ **faire**
   $dec\_y \leftarrow decompose(lvg)$
   $lg \leftarrow [compose(x, 1^\circ(dec\_y))|lg]$
   $lvg \leftarrow 2^\circ(dec\_y)$
  fin tant que
  retourner lg

Enfin l'opérateur $\oplus$ opère ainsi : la fonction *decompose* extrait de la liste lvg de paires (val, gardes), pour la variable y quantifiée existentiellement la plus externe du lieur, une paire constituée d'une liste de paires (val, liste des couples (valeur, arbre)) et d'une liste de paires (val, reste des gardes) ; la fonction *compose* construit pour y son ensemble de gardes en distribuant les arbres pour les différentes valeurs. Les fonctions 1° et 2° accèdent respectivement à la première et deuxième position d'un n-uplet ($n \geq 2$).

L'exemple suivant illustre le fonctionnement de l'algorithme *rec_comp*.

Exemple 6 *Nous calculons pour tout* $val_x, val_y \in \{0,1,2\}$ *la base* $B_{val_x val_y}$ *comme étant le résultat de l'appel :*

$$rec\_comp(\forall z \exists t, \{(x = (y * z) + t), (x = val_x), (y = val_y)\})$$

avec $z, t \in \{0,1,2\}$.

Nous obtenons les bases (selon $T^{val_x val_y}_{val_t}$*) :*

$$B_{00} = \langle \forall z \exists t | [[(0, T_0^{00} = \text{  })]] \rangle$$

$$B_{10} = \langle \forall z \exists t | [[(1, T_1^{10} = \text{  })]] \rangle$$

$$B_{20} = \langle \forall z \exists t | [[(2, T_2^{20} = \text{  })]] \rangle$$

$B_{21} = \langle \forall z \exists t \mid$
$[[(0, T_0^{21} = \text{  }), (1, T_1^{21} = \text{  }), (2, T_2^{21} = \text{  })]] \rangle$

et pour toute autre combinaison, $B_{val_x val_y} =$ *bottom_base.*

L'exemple suivant illustre le partage des arbres qu'opère l'opérateur $\oplus$ montrant ainsi la distribution des arbres ; cet exemple illustre aussi qu'une base est de taille linéaire par rapport à un arbre contenant uniquement l'ensemble des scénarios appartenant à des stratégies gagnantes.

Exemple 7 *Le combinateur existentiel de bases est appliquée lors de l'appel*

$$rec\_comp(\exists y \forall z \exists t, \{(x = (y * z) + t), (x = 2)\})$$

avec $y, z, t \in \{0,1,2\}$*, aux bases* B_{20}*,* B_{21} *et* B_{22} *qui représentent les bases pour les QCSP, respectivement,* $\forall z \exists t((x = (y * z) + t) \wedge (x = 2) \wedge (y = 0))$, $\forall z \exists t((x = (y * z) + t) \wedge (x = 2) \wedge (y = 1))$, $\forall z \exists t((x = (y * z) + t) \wedge (x = 2) \wedge (y = 2))$.

$\oplus_\exists(y, \forall z \exists t, [(0, B_{20}), (1, B_{21}), (2, B_{22})]) =$
$B_2 = \langle \exists y \forall z \exists t \, | [[(0, \square), (1, \square)],$
$[(0, \text{}), (1, \text{  }), (2, \text{})]] \rangle$

qui est une base compatible avec le QCSP

$$\exists y \forall z \exists t((x = (y * z) + t) \wedge (x = 2))$$

avec $y, z, t \in \{0,1,2\}$.

Le théorème suivant établit que non seulement l'algorithme *rec_comp* calcul à partir d'un QCSP une base compatible mais qu'en plus elle est optimale.

Théoreme 3 *Soit un QCSP* QC. $rec\_comp(Q, C)$ *retourne une base optimale et compatible avec le QCSP.*

4.2 Construction via un analyseur de trace

La section précédente a présenté une construction récursive de la base en considérant la définition inductive d'un QCSP et sa sémantique récursive mais en pratique il est souvent plus efficace de considérer, d'un côté, le solveur qui effectue la recherche et émet une trace d'exécution et, d'un autre côté, un analyseur de trace qui construit la base. Il y a deux raisons majeures à cette coopération entre deux processus via un flux :

- la gestion de la mémoire dans le solveur sous la forme d'une pile de retour-arrière qui doit, dans le cas d'un solveur incluant la construction de la base, stocker aussi l'état courant de celle-ci ;
- ternir compte des architectures modernes multi-cœurs.

L'analyseur de trace *it_comp* construit une base grâce à la trace d'exécution d'un solveur QCSP basé sur un algorithme de recherche. Cette trace est constituée d'une séquence de commandes de cinq types :

- `branche(x,val)` : la contrainte $(x = val)$ a été ajoutée à l'ensemble des contraintes, par branchement sur l'unique valeur possible du domaine lors de l'élimination du quantificateur pour les variables forcées par propagation ou par branchement lors d'un point de choix ;
- `echec` : le scénario partiel courant fait apparaître une contradiction dans l'ensemble des contraintes, celui-ci ne peut donc pas faire partie d'une stratégie gagnante ;
- `echec(`v_i`)` : le scénario courant $val_1, \ldots, val_{i-1}$ pour les variables $v_1, \ldots v_{i-1}$ est tel que le QCSP $\forall v_i q_{i+1} v_{i+1} \ldots q_n v_n (C \wedge \bigwedge_{1 \leq k < i} (v_k = val_k))$ est détecté comme sans stratégie gagnante ;
- `succes` : le scénario courant vérifie toutes les contraintes ;
- `succes_global` : une stratégie gagnante est calculée.

La trace t est un flux dans lequel le solveur enfile[5] les commandes et que l'analyseur défile grâce à la fonction $defile$ ou consulte grâce à la fonction *consulte*. La variable s (resp. p) représente les quantificateurs sur lesquels le solveur n'a pas encore (resp. a déjà) branché ; la variable sc représente le scénario courant ; la variable st représente le stock de scénarios gagnants ; la variable lg représente les gardes déjà calculées ; la variable c représente une commande ; la variable *etat* représente l'état de l'algorithme. L'état *descente* représente la descente dans l'arbre de recherche (i.e. la construction du scénario) et n'est modifié en état *backtrack* que lorsqu'un succès est rencontré (et le scénario est alors stocké dans la liste des scénarios gagnants) ou lorsqu'un échec est rencontré. L'état *backtrack* représente la remontée dans l'arbre de recherche à la recherche du point de choix précédent et n'est modifié que lorsque celui-ci est trouvé. Dans l'état de *descente* : la commande $branche(x, val)$ correspond à l'ajout de la contrainte $(x = val)$ dans l'ensemble des contraintes et entraîne l'ajout de val dans le scénario et du glissement du quantificateur de s à p. Dans l'état de *backtrack* : la commande $branche(x, val)$ permet de retrouver le point de choix en faisant glisser les quantificateurs de p à s jusqu'à ce que la variable de branchement corresponde à la variable associée au quantificateur ; la commande *succes_global* correspond à l'insertion d'une stratégie gagnante dans la liste des gardes partielles via la fonction *insere* ; la commande $echec(u)$ élimine des scénarios gagnants ceux qui associent, pour le même scénario partiel que le courant, d'autres valeurs à la variable universelle u. Les fonctions 3° et 4° accèdent respectivement à la troisième et quatrième position d'un 4-uplet.

5. les fonctions $defile$, $enfile$ et *consulte* sont les opérations classiques sur une file.

Algorithme 5 *it_comp*

Entrée: Q : un lieur d'un QCSP
Entrée: t : une trace d'exécution d'un solveur QCSP
Sortie: une base

$etat = descente; s = Q; p = \varepsilon; sc = []; st = []$
$lg = gardes\_vides(Q)$
tant que $!vide(t)$ **faire**
 $c \leftarrow consulte(t)$
 si $etat = descente$ **alors**
  $defile(t)$
  selon c **faire**
   cas $branche(x, val)$:
   $s \leftarrow queue(s); p \leftarrow [tete(s)|p]; sc \leftarrow [val|sc]$
   cas $succes$:
   $etat = backtrack; st \leftarrow [sc|st]$
   cas $echec$:
   $etat = backtrack$
  fin selon
 sinon
  selon c **faire**
   cas $branche(x, val)$:
   $p \leftarrow queue(p); s \leftarrow [tete(p)|s]$
   si $x = 1^\circ(tete(sc))$ **alors**
    $etat = descente$
   fin si
   $sc \leftarrow queue(sc)$
   cas $succes\_global$:
   $defile(t)$
   $lg \leftarrow insere(st, lg)$
   cas $echec(u)$:
   $defile(t)$
   $q \leftarrow destocke(u, s, p, sc, st)$
   $s \leftarrow 1^\circ(q); p \leftarrow 2^\circ(q); sc \leftarrow 3^\circ(q); st \leftarrow 4^\circ(q)$
  fin selon
 fin si
fin tant que
retourner $\langle Q \mid lg \rangle$

Exemple 8 *La figure 3 présente les ensembles de gardes G'_x, G'_y et G'_t de la base $B' = \langle \exists x \exists y \forall z \exists t \mid [G'_x, G'_y, G'_t] \rangle$, qui est calculée par l'analyseur de trace it_comp après avoir intégré les trois premières stratégies gagnantes pour le QCSP : $\exists x \exists y \forall z \exists t (x = (y * z) + t)$ avec $x, y, z, t \in \{0, 1, 2\}$.*

L'analyseur de trace it_comp est alors dans l'état backtrack avec les quantificateurs sur lesquels le solveur n'a pas encore branché $s = []$ (respectivement, a déjà branché $p = [\exists t, \forall z, \exists y, \exists x]$), le scénario courant

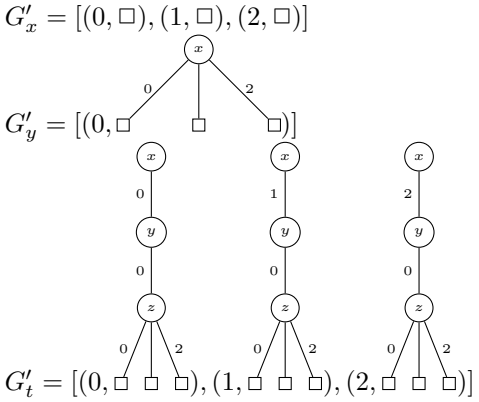
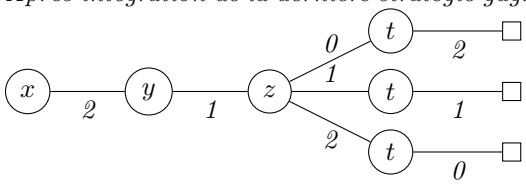


FIGURE 3 – Ensembles de gardes pour une base intermédaire.

inversé $[2,2,0,2]$ [6] *et la trace :*

$[branche(y,1), branch(z,0), branche(t,0), echec,$
$branche(t,1), echec, branche(t,2), succes,$
$branche(z,1), branche(t,0), echec, branche(t,1),$
$succes, branche(t,2), echec, branche(z,2),$
$branche(t,0), succes, branche(t,1), echec,$
$branche(t,2), echec, succes\_global, branche(y,2),$
$branche(z,0), branche(t,0), echec, branche(t,1),$
$echec, branche(t,2), succes, branche(z,1),$
$branche(t,0), succes, branche(t,1), echec,$
$branche(t,2), echec, branche(z,2), branche(t,0),$
$echec, branche(t,1), echec, branche(t,2),$
$echec, echec(z)]$

Après intégration de la dernière stratégie gagnante :



nous obtenons la base de la figure 2 de l'exemple 5.

Le théorème suivant établit que non seulement l'algorithme *it_comp* calcule à partir d'un QCSP une base compatible mais qu'en plus elle est optimale.

Théoreme 4 *Soit un QCSP QC et T la trace obtenue pour un solveur QCSP basé sur un algorithme de recherche. it_comp(Q,T) retourne une base optimale et compatible avec le QCSP.*

4.3 Complémentarité des deux approches

Les deux approches décrites dans les deux sous-sections précédentes sont complémentaires dans une architecture multicœurs/multiprocessus : à des couples

6. i.e. il faut lire $(t,2),(z,2),(y,0),(x,2)$.

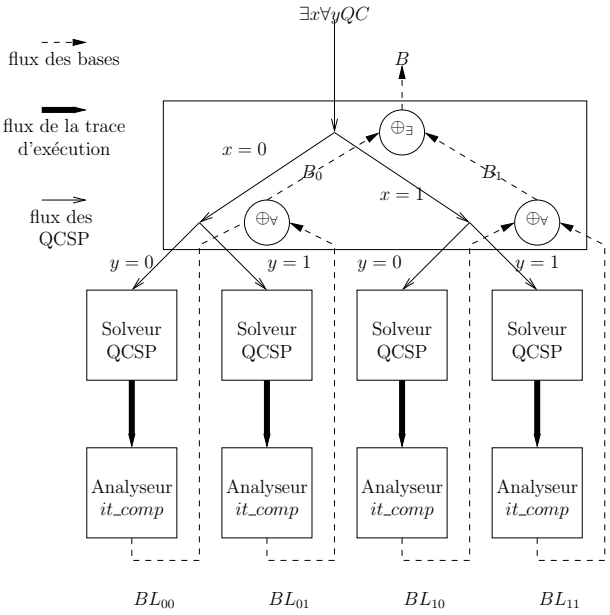


FIGURE 4 – Exemple d'architecture à 9 processus combinant les opérateurs $\oplus_\exists$ et $\oplus_\forall$ et l'analyseur de trace *it_comp*.

(algorithme de recherche pour QCSP, analyseur de trace *it_comp*) travaillant en parallèle, la tâche est confiée de calculer des bases pour des sous-problèmes tandis qu'un serveur combine ces bases grâce aux opérateurs $\oplus_\exists$ et $\oplus_\forall$.

Exemple 9 *La figure 4 présente pour un QCSP $\exists x \forall y QC$ avec $x,y \in \{0,1\}$, un exemple d'architecture avec un serveur réalisant la recherche (ici les branchements sur les valeurs des domaines des variables x et y) et appliquant aux bases B_{00}, B_{01} B_{10} et B_{11} l'opérateur $\oplus_\forall$ puis aux bases résultats B_0 et B_1 l'opérateur $\oplus_\exists$ pour obtenir une base finale B compatible avec le QCSP et optimale ; les bases B_{00}, B_{01} B_{10} et B_{11} ayant été obtenues dans quatre processus différents par une collaboration entre un algorithme de recherche et l'analyseur de trace it_comp pour les QCSP, respectivement, $Q(C \wedge (x=0) \wedge (y=0))$, $Q(C \wedge (x=0) \wedge (y=1))$, $Q(C \wedge (x=1) \wedge (y=0))$ et $Q(C \wedge (x=1) \wedge (y=1))$.*

5 Conclusion

Nous avons proposé dans cet article un cadre formel pour la compilation des QCSP : les bases. Grâce à nos deux algorithmes *rec_comp* et *it_comp*, nous sommes à même de nous adapter à quasiment toute architecture basée sur un algorithme de recherche. Ces algorithmes permettent pour des QCSP de générer des bases optimales qui leur soient compa-

tibles. Nous disposons d'implantations en Prolog pour les algorithmes décrits dans cet article disponibles à l'adresse `http://www.info.univ-angers.fr/pub/stephan/Research/Download.html` et projetons de l'insérer dans sa version itérative *it_comp* à notre solveur QCSP développé dans l'environnement générique pour le développement de systèmes de contraintes Gecode [14].

Dans le cadre d'un algorithme de décision pour les QCSP, lorsque le solveur répond qu'il y a ou n'y a pas de stratégie gagnante, rien ne permet de le vérifier. Un certificat est une information, qui peut être une stratégie gagnante mais pas nécessairement, qui permet de vérifier que l'algorithme à bien décider du problème. Nous n'avons pas pu, faute de place, traiter les certificats pour les QCSP : notre formalisme les inclut comme un cas particulier ; l'interprétation des arbres sous la forme de contraintes tables permet de vérifier un tel certificat vis-à-vis d'un QCSP par la résolution d'un problème co-NP-complet (rejoignant ici la complexité de la vérification d'une politique pour une QBF [3]).

Références

[1] F. Bacchus and K. Stergiou. Solution directed backjumping for qcsp. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, pages 148–163, 2007.

[2] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E.Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of the international conference on Computer-aided design (ICCAD'93)*, 1993.

[3] M. Benedetti. Extracting Certificates from Quantified Boolean Formulas. In *Proceedings of 9th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 47–53, 2005.

[4] M. Benedetti, A. Lallouet, and J. Vautard. Reusing csp propagators for qcsps. In *Recent Advances in Constraints, 11th Annual ERCIM International, Workshop on Constraint Solving and Contraint Logic Programming (CSCLP'06)*, pages 63–77, 2006.

[5] L. Bordeaux and E. Monfroy. Beyond NP : Arc-Consistency for Quantified Constraints. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP'02)*, pages 371–386, 2002.

[6] M. Cadoli and F.M. Donini. A survey on knowledge compilation. *AI Communications*, 10(3-4) :137–150, 1997.

[7] H. Chen. The Computational Complexity of Quantified Constraint Satisfaction, PhD thesis, cornell university, 2004.

[8] S. Coste-Marquis, D. Le Berre, F. Letombe, and P. Marquis. Complexity Results for Quantified Boolean Formulae Based on Complete Propositional Languages. *Journal on Satisfiability, Boolean Modeling and Computation*, 1 :61–88, 2006.

[9] A. Darwiche and P. Marquis. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research*, 17 :229–264, 2002.

[10] H. Fargier and P. Marquis. On the Use of Partially Ordered Decision Graphs in Knowledge Compilation and Quantified Boolean Formulae. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI'06)*, 2006.

[11] I. Gent, P. Nightingale, and K. Stergiou. Qcsp-solve : A solver for quantified constraint satisfaction problems. In *Proceedings of 9th International Joint Conference on Artificial Intelligence (IJCAI'05)*, 2005.

[12] I.P. Gent, P. Nightingale, and A. Rowley. Encoding Quantified CSPs as Quantified Boolean Formulae. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, pages 176–180, 2004.

[13] A.K. Mackworth. Consistency in networks of relations. In *Artificial Intelligence*, volume 8, pages 99–118, 1977.

[14] C. Schulte and G. Tack. Views and Iterators for Generic Constraint Implementations. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, pages 817–821, 2005.

[15] I. Stéphan and B. Da Mota. Base littérale et certificat pour les formules booléennes quantifiées. In *Actes des Troisièmes Journées Francophones de Programmation par Contraintes (JFPC'08)*, pages 307–316, 2008.

[16] I. Stéphan and B. Da Mota. A unified framework for Certificate and Compilation for QBF. In *Proceedings of the 3rd Indian Conference on Logic and its Applications*, pages 210–223, 2009.

[17] G. Verger and C. Bessière. BlockSolve : une approche bottom-up des QCSP. In *Actes des Deuxièmes Journées Francophones de Programmation par Contraintes (JFPC'06)*, pages 337–345, 2006.