Quantified Constraint Optimization*

Marco Benedetti, Arnaud Lallouet, and Jérémie Vautard

Université d'Orléans — LIFO
BP 6759 — F-45067 Orléans cedex 2
{Marco.Benedetti,Arnaud.Lallouet,Jeremie.Vautard}@univ-orleans.fr

Abstract. Solutions to valid Quantified Constraint Satisfaction Problems (QCSPs) are called winning strategies and represent possible ways in which the existential player can react to the moves of the universal one to "win the game". However, different winning strategies are not necessarily equivalent: some may be preferred to others. We define Quantified Constraint Optimization Problems (QCOP) as a framework which allows both to formally express preferences over QCSP strategies, and to solve the related optimization problem. We present examples and some experimental results. We also discuss how this framework relates to other formalisms for hierarchical decision modeling known as von Stackelberg games and bilevel (and multilevel) programming.

1 Introduction

QCSP (Quantified Constraint Satisfaction Problems) is a constraint-based framework used to model several problems that go beyond classical CSP, such as those involving some degree of uncertainty in the state of the modeled reality, and those structured as game playing [1,2,3] or adversary problems [4], such as conformant planning, model checking, testing, and robust scheduling [5], to name a few.

In QCSP variables may be universally quantified over their domains. Such universal quantification is crucial while modeling, for example, the behavior of a hostile adversary or some potentially harmful uncertainty about the state of the environment. This expressive power comes at a cost: While CSP is solved by just exhibiting values for its (existentially quantified) variables such that all the constraints are satisfied, a QCSP is solved by exhibiting *winning strategies*. A strategy is a set of functions that compute the values of each (existentially quantified) variable in the problem as a function of all the relevant (universally quantified) variables. A *winning strategy* is a strategy that, given whatever assignment to the universal variables, manages to satisfy all the constraints by the values of the existential variables: then, a QCSP is true if it has at least one winning strategy. Strategies, unlike satisfying CSP assignments, are not always docile objects: their worst-case size is exponential in the number of variables.

Given a true QCSP instance, are all its winning strategies equally desirable? It turns out that some strategies should be preferred over others, despite being

* This work is supported by the project ANR-06-BLAN-0383.

P.J. Stuckey (Ed.): CP 2008, LNCS 5202, pp. 463–477, 2008.

equally "winning". The contribution of this paper is to present a framework, called Quantified Constraint Optimization (QCOP$^+$), to express preferences over strategies, and a reasoning engine that solves the resulting optimization problem.

In CSP we express preferences over the set of solutions by means of an objective function, and solve the related optimization problem by determining the satisfying assignment(s) that maximize(s)/minimize(s) such objective. Similarly, in a game-like scenario modeled in QCSP we could be interested, for example, in playing the strategy that gives the earliest win, or in selecting strategies with features that cannot be enforced at the level of the constraint language. As an example, suppose that we face a game-like situation in which strategies to force the opponent to a tie exist, though we cannot defeat him if he plays perfectly. Out of all the strategies that prevent the opponent from winning, we prefer those leading to our win in case the opponent plays less than perfectly. This induces a preference over the space of winning strategies which cannot be modeled in plain QCSP (any attempt to require a strict defeat of the opponent by additional constraints would result in a false instance, as the opponent cannot be overcome in general). QCOP$^+$ is the perfect framework for modeling similar situations.

The QCOP$^+$ language we introduce is based on QCSP$^+$ [3] and extends it by providing means to define *compositional objective functions* built along the quantification structure of the problem. With each universal quantification we associate some *aggregate function*, while *optimization functions* (e.g., minimization, maximization) are associated with existential levels. Let us consider, for example, the QCOP$^+$ in Figure 1, in which the domains are numerical. We associate the components of the optimization function with a specific scope by indenting them at the same level as the quantifier they refer to. If we momentarily disregard lines $(5-7)$, we recognize a standard QCSP$^+$ instance $P = \exists X \in D_X[C_1]\ \forall Y \in D_Y[C_2]\ \exists Z \in D_Z[C_3].\ C$, where the conditions C_1 to C_3 are used to restrict dynamically the possible values a variable may assume. Let $W \subseteq S_X \times S_Z$ be the set of winning strategies for such problem, where S_X is the space of (constant) functions onto D_X and S_Z is the space of functions $s_Z : D_Y \mapsto D_Z$, and let W_X denote the set $\{s_X \in S_X : \exists s_Z\ \langle s_X, s_Z\rangle \in W\}$. Then, our sample QCOP$^+$ instance asks to identify the subset $W' \subseteq W$ of winning strategies which, beyond satisfying P, optimize the objective function:

(1) $\exists X \in D_X\ .\ [C_1(X)]$
(2) $\quad \forall Y \in D_Y\ .\ [C_2(X,Y)]$
(3) $\quad\quad \exists Z \in D_Z\ .\ [C_3(X,Y,Z)]$
(4) $\quad\quad\quad C(X,Y,Z)$
(5) $\quad\quad \min(Z)$
(6) $\quad k : sum(Z)$
(7) $\max(k)$

Fig. 1. Example of QCOP$^+$

$$\max_{s_X \in W_X} [\sum_{Y \in D_Y} (\min_{\langle s_X, s_Z\rangle \in W} s_Z(Y))]$$

Any QCOP$^+$ instance is thus composed of two parts: an initial QCSP$^+$ portion which identifies candidate winning strategies, followed by a quantifier-by-quantifier specification of an objective function meant to describe optimal candidates. These two parts belong to conceptually different languages and manipulate

different domains: the first part deals with truth assignments to the problem variables, the second one is concerned with strategies and sub-strategies.

By mixing them in a single specification we obtain several benefits: we make as easy as possible the task of declaring (complex) preferences over (complex) strategies; we keep all the information about the quantified optimization problem in a compact specification; most importantly, as we shall see, we give the solver the possibility to exclude on-the-fly partially-formed sub-optimal candidates, in the spirit of branch and bound algorithms.

We have found no previous account for a general notion of optimization in the QCSP literature. However, this kind of problems has been studied since the 70's in mathematical programming under he name of *bilevel* (or *multi-level*) programming [6], a.k.a. "mathematical programs with optimization problems in the constraints". Bilevel programs are used for solving decision problems of the form of Stackelberg games, which is a model of oligopoly in game theory [7]. In this kind of problems, there are two actors who perform decisions sequentially but have no control on each other. The first one to act is called "leader"; the second one, called "follower", uses the leader decisions to adapt her own ones towards her objective. The key issue here is that the leader and the follower have different objective functions, that may be conflicting. For example, the leader can be a government agency which divides an amount of money among several entities which are free to use their amount for their own purpose. The following example, taken from [8], shows that conflicting objectives can lead to a non-optimal equilibrium. In the situation depicted in Figure 2, the choice of (x_1, y_1) which would be optimal without the follower becomes considerably sub-optimal with her response (x_2, y_1). The equilibrium x^* is depicted in Figure 3, where it can be noticed that dominating solutions exist, for both the leader and the follower, which can never be reached without consensus. We refer to [9] for an extensive survey of bilevel programming. The name multi-level programming applies when more than two levels of hierarchical decisions are involved.

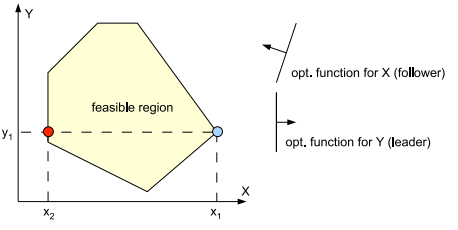

Fig. 2. Optimum for the leader alone and response of the follower

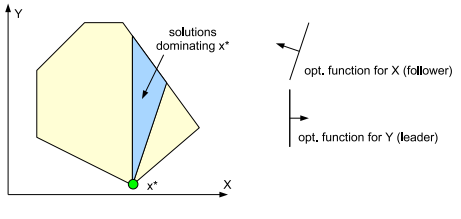

Fig. 3. Optimal equilibrium

The QCOP$^+$ framework has been prototyped in the solver QeCode [10] based on Gecode [11]. In addition to optimization, it includes strategy extraction (useful also for classical QCSP$^+$, where a simple *true/false* answer is often insufficient). The extraction of strategies has been introduced in [12] in the context of QBF.

The paper presents QCSP, QCSP$^+$ and then QCOP$^+$ and their evaluation. The search algorithm is presented and its branch and bound variant studied. Finally some examples of bilevel problems are presented.

2 QCSP

Notations. Let V be a set of variables and $D = (D_X)_{X \in V}$ be the family of their domains. We recall that a family is a function from an index set to a set. For $W \subseteq V$, we denote by D^W the set of tuples on W, namely $\Pi_{X \in W} D_X$. Projection of a tuple (or a set of tuples) on a variable (or a set of variables) is denoted by $|$. For example, for $t \in D^V$, $t|_W = (t_X)_{X \in W}$ and for $E \subseteq D^V$, $E|_W = \{t|_W \mid t \in E\}$. For $W, U \subseteq V$, the join of $A \subseteq D^W$ and $B \subseteq D^U$ is $A \bowtie B = \{t \in D^{A \cup B} \mid t|_W \in A \ \wedge \ t|_U \in B\}$. A sequence is a family indexed by a prefix of $\mathbb{N}$. We denote by $|$ the sequence constructor and by $[]$ the empty sequence. We use $a?b{:}c$ to denote *if a then b else c*.

Constraints and CSPs. A *constraint* $c = (W, T)$ is a couple composed of a subset $W \subseteq V$ of variables and a relation $T \subseteq D^W$ (W and T are also respectively noted $var(c)$ and $sol(c)$. An empty constraint such that $sol(c) = \emptyset$ is *false* and a full constraint (which does no constrain the variables) is such that $sol(c) = D^W$. When $W = \emptyset$, only these two constraints exist: $(\emptyset, \emptyset)$ which has value *false* and $(\emptyset, ())$ which has value *true*.

A *Constraint Satisfaction Problem* (or CSP) is a set of constraints. We denote by $var(C) = \bigcup_{c \in C} var(c)$ its set of variables and by $sol(C) = \bowtie_{c \in C} sol(c)$ its set of solutions. The empty CSP which contains no constraint is *true* and will be denoted by $\top$ while any CSP which contains a false constraint is *false* and denoted by $\bot$.

Prefix and QCSP. A *quantified set of variables*, or *qset* is a couple (q, W) where $q \in \{\exists, \forall\}$ is a quantifier and $W \subseteq V$.

Definition 1 (Prefix). *A* prefix P *is a sequence of qsets* $[(q_0, W_0), \ldots, (q_{n-1}, W_{n-1})]$ *such that* $i \neq j \Rightarrow W_i \cap W_j = \emptyset$.

We denote by $P|_W$ the prefix P restricted to the variables of a set W. A variable X is *declared* in a qset W_i if $X \in W_i$. A QCSP is defined by adding a CSP to a prefix:

Definition 2 (QCSP). *A* Quantified CSP*, or* QCSP *is a couple* (P, G) *where* P *is a prefix and* G *is a CSP called* goal.

Let $P = [(q_0, W_0), \ldots, (q_{n-1}, W_{n-1})]$ be a prefix. We define the following notations. First, let $range(P) = [0..n]$. For all i in $range(P)$, let $var_i(P) = W_i$ be the set of variables at index i, let $before_i(P) = \bigcup_{j \leq i} var_j(P)$ (resp. $after_i(P) = before_n(P) \backslash before_i(P)$) be the set of all variables defined before (resp. after) the index i. We also need to access to the index of the next universal block $nu_i(P)$ located after an index i. We define $nu_i(P) = min_{j>i}\{j \mid q_j = \forall\}$ if such an index exists, and n otherwise. These notions are naturally extended for QCSP

$Q = (P, G)$ in a straightforward way. Moreover, we have $\mathit{prefix}(Q) = P$ and $\mathit{goal}(Q) = G$. The QCSP is *closed* if $\mathit{var}(G) = \mathit{before}_n(Q)$, i.e. all variables mentioned in the goal are explicitly quantified. In the sequel, we only consider closed QCSP.

Example 3 (QCSP). The formula:

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

is represented by the following QCSP, in which the domains attached to the variables are not mentionned:

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

Thus, $\mathit{prefix}(Q) = [(\exists, X), (\forall, Y), (\exists, Z)]$, $\mathit{goal}(Q) = \{X + Y = Z\}$, $\mathit{range}(Q) = [1..3]$, $\mathit{var}_1(Q) = \{X\}$, $\mathit{before}_2(Q) = \{X, Y\}$, $\mathit{after}_2(Q) = \{Z\}$. □

Strategy and scenario. A solution, called *strategy*, is intuitively the way the existential player react to every possible move of the the universal player. It is interesting to note that a strategy is a syntactic object that does not depend on a notion of validity. It is just a possible way to play the game as if there is no rule. As a consequence, it can be defined for a prefix only. In [13], a strategy was defined as a family of (Skolem) functions that give a value to an existential variable as a function of its preceding universal ones. For the purpose of this set-theoretic exposition, we rather define it in extension, as a set of tuples. Each of these tuples is a *scenario*, i.e. a possible way the game is played. Here follows the inductive definition of the set of strategies for a given prefix:

Definition 4 (Set of strategies). *The* set of strategies $\mathit{Strat}(P)$ *for a prefix* $P = [(q_0, W_0), \ldots, (q_{n-1}, W_{n-1})]$ *is defined inductively as follows:*

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

The set of strategies for a prefix beginning with an universal variable is defined as follows: we build, for a tuple $t \in D^W$, the set $\{ t \bowtie s' \mid s' \in \mathit{Strat}(P') \}$ of all strategies beginning by t. Then we take the Cartesian product $\Pi_{t \in D^W} (\{ t \bowtie s' \mid s' \in \mathit{Strat}(P') \})$ of all these sets. Each tuple α of this Cartesian product has as value a strategy beginning by a tuple t for every $t \in D^W$. Such tuple α is also a function that associates to each tuple of D^W a strategy, which is a set of tuples. The union of the strategies of the image set $\alpha(D^W)$ of this function is a new strategy which contains a sub-strategy for each $t \in D^W$. The set of strategies for the prefix is the set of all strategies constructed by all tuples α.

Semantics of QCSP. A strategy is a *winning strategy* if all of its scenarios satisfy the goal:

Definition 5 (Winning Strategy for QCSP). *A strategy* s *is a* winning strategy *iff* $s|_{\mathit{var}(G)} \subseteq \mathit{sol}(G)$.

We denote by $\textsc{win}(Q)$ the set of all winning strategies of Q.

Definition 6 (Semantics of QCSP). *The* semantics $[\![Q]\!]$ *of a QCSP* Q *is:*

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

This notion of solution generalizes exactly the classical notion of solution of CSP: a QCSP is true if it has a winning strategy. Other weaker notions have been proposed. The notion of *outcome*, which is the set of scenarios of all winning strategies, has been used as a notion of solution for QCSP in [13] to model filtering.

Example 7. Consider the following QCSPs:

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

With the tuples defined for the values of x, y and z respectively, we have:

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

QCSP$^+$. Introducing restricted quantification in QCSPs means changing the nature of the prefix. In addition to a quantifier and a set of variables, each scope includes a CSP whose solutions define the allowed values for the variables of the current qset. QCSP$^+$ have been introduced in [3] mainly for modeling purposes. A *restricted quantified set of variables*, or *rqset* is a triple (q, W, C) where $q \in \{\exists, \forall\}$ is a quantifier, $W \subseteq V$ and C is a CSP. The intended meaning is to restrict the possible values of the variables of W to those which satisfy the CSP C. We extend the notion of prefix to rqsets. In particular, it is still required that $i \neq j \Rightarrow W_i \cap W_j = \emptyset$.

Definition 8 (QCSP$^+$). *A QCSP$^+$ is a couple* $Q = (P, G)$ *where* P *is a prefix of rqsets such that* $var(C_i) \cap after_i(Q) = \emptyset$ *and* G *is a goal CSP.*

A QCSP$^+$ $Q = (P, G)$ is closed if $\forall i \in range(P), var(C_i) \subseteq before_i(Q)$ and $var(G) \subseteq before_n(Q)$. It is easy to notice that a standard QCSP is a QCSP$^+$ for which $\forall i \in range(P), C_i = \emptyset$. The definition of strategy for a QCSP$^+$ is the same as for a QCSP. But being a winning strategy is different. A winning strategy is a strategy for which all possible moves for the universal player end in a winning scenario. This can happen, like in a classical QCSP, when all constraints of the goal and of all restrictions are satisfied because all implications and conjunctions are valid. But, it can also happen when one left-hand side of an implication is contradicted. Then this scenario is valid whatever happens in the remaining assignments of variables after the contradicted rqset's CSP. The set of winning strategies of a QCSP$^+$ can be defined recursively as follows:

Definition 9 (Set of winning strategies for a QCSP$^+$). *Let Q be a QCSP$^+$. The* set of winning strategies $\text{WIN}(Q)$ *is defined by:*

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

This definition is similar to the definition of the set of strategies for a prefix. However, classical winning sub-strategies are not the only ones to take into account: A strategy can be winning at a universal level if it contradicts the related restriction. It follows that any sub-strategy can be freely glued, whatever its winning status. To reason on all the CSP restrictions at once, a kind of propagation called cascade propagation is introduced in [3].

3 Optimization

QCOP$^+$ is formed after QCSP$^+$ by adding preferences and aggregates to the rqsets. Let $\mathcal{A}$ be a set of aggregate names and $\mathcal{F}$ be a set of aggregate functions. An aggregate function is defined by an associative function on a multiset of values and a neutral element 0_f which indicates the value of $f(\{\!\{\}\!\})$. Possible functions are *sum, product, average, standard deviation, median, count*, etc. If the function is associative and commutative, it can be evaluated using an accumulator initialized to 0_f and the evaluation could be parallelized. An *aggregate* is an atom of the form $a : f(X)$ where $a \in \mathcal{A}, f \in \mathcal{F}$ and $X \in V \cup \mathcal{A}$. We call $names(A)$ the set of aggregate names of a set of aggregates A. An aggregate name has the same status as a variable, except that it cannot be part of a constraint. An *optimization condition* is an atom of the form $\min(X), \max(X)$ where $X \in V \cup \mathcal{A}$ or the atom *any*. An atom $\min(X)$ indicates that the user is interested in strategies that minimize this value and not in the other ones, while *any* simply indicates she does not care about the returned strategy. It is only needed to define minimization since $\max(X)$ is a syntactic sugar for $\min(-X)$.

Definition 10 (Orqset). *An* $\exists$-orqset *is a 4-uple* $(\exists, W, C, o)$ *where* $(\exists, W, C)$ *is a rqset and o is an optimization condition. A* $\forall$-orqset *is a 4-uple* $(\forall, W, C, A)$ *where* $(\exists, W, C)$ *is a rqset and A is a set of aggregates. An* orqset *is either an* $\exists$*-orqset or a* $\forall$*-orqset.*

The notion of prefix and all adjoin notations defined in notation 2 are extended to a sequence of orqsets. There is a restriction on the variables which can appear in an optimization condition or an aggregate. Actually, it should be ensured that the variable to be optimized will have an unique value in the current strategy. This is the case if this variable is not in the scope of an universal quantifier located after the optimization condition. However, it can be an aggregate of the next universal block since it will also have an unique value. The same holds for the variable of an aggregate: it can belong to the set of variables of any existential scope between the aggregate declaration and the next universal block's aggregates.

Definition 11 (QCOP and QCOP$^+$). *A QCOP$^+$ is a couple (P, G) where G is a CSP and $P = [orq_0, \ldots, orq_{n-1}]$ is a prefix of orqsets such that $\forall i \in range(P)$, with $k = nu_i(P)$:*

- *if $orq_i = (\exists, W, C, o)$ with $o = min(X)$ or $o = max(X)$, then we must have $X \in before_{k-1}(P) \cup (k < n \ ? \ names(A_k) : \emptyset)$*
- *if $orq_i = (\forall, W, C, A)$, then for all $a : f(X)$ in A, we must have $X \in before_{k-1}(P) \cup (k < n \ ? \ names(A_k) : \emptyset)$*

A QCOP is a QCOP$^+$ in which no orqset has restrictions.

The semantics of a QCOP$^+$ is defined as a set of strategies which include the computation of the aggregates and which respect the optimization conditions. We first define the function *val* which computes the value of an aggregate $a{:}f(X)$ for a strategy s. We have $val(a{:}f(X), s) = f(\{\!\{t|_X \mid t \in s\}\!\})$.

Definition 12 (Semantics of QCOP$^+$). *The semantics of a QCOP$^+$ is a set of strategies defined as follows:*

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

After the aggregates are evaluated, their value are glued to the scenarios of the strategy and they appear as if they were existential variables of the preceding level. As for CSP which can have multiple optimal solutions, a QCOP$^+$ may have multiple optimal strategies. It can happen with the use of *any*, but also when multiple strategies have the same optimal value. They may differ a lot on subsequent optimal values found in sub-strategies. However, the search algorithm described in the next section returns one of these optimal strategies only.

4 Algorithms

This section presents a search algorithm to evaluate QCOP$^+$ and its version based on branch and bound. It is implemented in the solver QeCode [10] based on Gecode [11]. The solving technique is based on the QCSP$^+$ search procedure that recursively explores the quantified structure. A mechanism for strategy extraction and its recording in a tree is implemented. This feature also benefits to QCSP$^+$ because in many cases the user is interested not only in the decision problem but also in the way the game can be played. An explicit representation of strategies—called certificate in [12]—has numerous applications, the first one being to be able to verify the solution in a solver-independent way. In the current

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

Procedure
Solve_e ([(∃, W, C, min(X))|P'], G)
  BEST_STR := null
  BEST_Xvalue := +∞
  for all t ∈ D^W s.t. t is a solution of C
  do
    CUR_STR := Solve( (P', G)[W ← t] )
    if CUR_STR ≠ 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 ([(∀, W, C, A)|P'], G)
  for all ∀a:f(X) ∈ A do
    VAL_a := ∅
  end for
  STR := ∅
  for all t ∈ D^W s.t. t is a solution of C
  do
    CUR_STR := Solve( (P', G)[W ← t] )
    if CUR_STR = null then
      return null
    else
      for all a:f(X) ∈ A do
        VAL_a := VAL_a ⊎
          {{ CUR_STR|_X }}
      end for
      STR := STR ⋃ CUR_STR
    end if
  end for
  return tree((f(VAL_a))_{a:f(X)∈A}, STR)
```

Fig. 4. Search procedure

prototype implementation, the tree is recorded without compression, and this could eventually put limits to the size of the examples that can be handled.

The main search procedure is composed of two mutually recursive evaluation functions, one for an ∃-orqset and one for a ∀-orqset. They return a strategy described by a tree which can either be the empty tree *null* or *tree(a,B)* where a is a tuple and B a set of trees. The general algorithm, the algorithm for a minimization condition in an ∃-orqset and the one for a ∀-orqset are given in Figure 4. For an ∃-orqset, the function maintains the best strategy found so far *BEST_STR* and returns it, or null if the orqset is failed. All strategies are successively explored and compared on their X value. The max and *any* aggregates are defined similarly. Adding branch and bound to this procedure can be done simply by adding the constraint $X <$ *BEST_Xvalue* (resp. $>$) to the rest of the minimization (resp. maximization) problem. This can be seen as an adaptation of the algorithm of [14] for which the lower/upper bounds are directed by the optimization condition and associated to their own optimization variable instead of to the whole problem. Once a solution has been found, the algorithm for a ∀-orqset first evaluates the sub-strategies for every universal tuple. For each of them it computes the set of aggregates. Then it collects all of them in a set *STR* and returns it at the end.

Branch and Bound. Interestingly, the branch and bound algorithm may be incorrect in the case of *overlapping* optimization conditions. This happens if there

exists two orqsets $orq_i = (\exists, W_i, C_i, min(X))$ with $X \in W_k$ and $orq_j = (\exists, W_j, C_j, min(Y))$ with $Y \in W_l$ such that $i < j < k$. Any number of condition *any* may appear in between.

Example 13. A sample problem incorrect for branch and bound is in Figure 5.

Suppose there exists three strategies $s_0 = \{(X_0, Y_0, A_0, B_0)\}$, $s_1 = \{(X_1, Y_1, A_1, B_1)\}$ and $s_2 = \{(X_1, Y_2, A_2, B_2)\}$ such that $A_1 > A_0$, $A_2 < A_0$ and $B_1 < B_2$. Having found the strategy s_0, the constraint $A < A_0$ is added to the search of subsequent strategies. Thus, s_1 is cut. We find s_2 which has a better value A_2 for A and the optimal strategy is s_2. Without branch and bound, optimization at the level of Y would have preferred strategy s_1 because of its better value on B and would have returned the value A_1. The best strategy at the upper level would have been s_0.

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

Fig. 5. Incorrect B&B

Proposition 14. *Branch and bound is correct if optimization conditions are non-overlapping.*

Proof. With the same notations as above, conditions are non-overlaping if $k \leq j$. Then it is ensured that any branch cut by B&B will be cut before the level of Y will be reached, hence only strategies worse for X will be cut.

5 Examples

In this section, we give several examples of use of quantified constraint optimization, ranging from toy examples to real-world problems taken from the bilevel programming literature. In order to give a readable presentation of the examples, we use a pseudo-code syntax in which the aggregates/optimization part is placed at the end. In a similar way, we allow the use of constants and arrays. For example, the following QCOP$^+$ which returns a strategy in which $X = 0$ if the sum of odd indices of the array A is less than the sum of its even indices and $X = 1$ otherwise is depicted on the right as pseudo-code:

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

```
const A[0..9]
\exists X in {0..1}
|  \forall i in {0..9} [i mod 2 = X]
|  |  \exists Z in {0..+oo}
|  |  |  Z = A[i]
|  |  any
|  s:sum(Z)
min(s)
```

Minimax in adversary scheduling. Often, the objectives of the existential and universal players conflict completely. This situation can be dealt with by a classical minimax algorithm (and in this case the branch and bound implements alpha-beta pruning). We illustrate this case by an extension of the adversary scheduling example introduced in [4]. In this problem, two opponents are involved: the *scheduler* tries to build a schedule that satisfies all the (temporal and resource) constraints, while the *adversary* tries to prevent the formation of a valid schedule by inflicting some (limited) deterioration on the problem setting. In the QCSP$^+$ version, the scheduler was trying to build a schedule such that the ending date was below a given threshold. In QCOP$^+$ it is possible to capture the more realistic variant in which the scheduler aims to minimize the ending time of the robust schedule, while the adversary tries to maximize the same ending time. Let us consider an example with three activities a_1, a_2, a_3 and a resource r. An activity a_i has a starting date of s_i, a duration of d_i and requires c_i units of the resource r of maximal capacity 5. The precedence is $a_1 \prec a_2$ and the data are $d_1 = 1, d_2 = 2, d_3 = 3, c_1 = 3, c_2 = 2$ and $c_3 = 1$. The adversary is able to add one unit to the resource consumption of at most two activities. We add a fictitious activity *end* whose purpose is to minimize the length of the schedule. The model in QCOP$^+$ is as follows:

```
const d[1..3], c[1..n]
\exists k1 in {0,1}, k2 in {0,1}, k3 in {0,1}
|  [k1+k2+k3 =< 2]
|  \exists S1 in D1, S2 in D2, S3 in D3, Send in Dend,
|  |       c'1 in Dc1, c'2 in Dc2, c'3 in 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)
```

Since there are only existential variables, the strategy is reduced to a single branch which gives the best attack and the corresponding scheduler response.

Network links pricing. Here is an example from the telecom industry, taken from [15]. The problem is to set a tariff on some network links in a way that maximizes the profit of the owner of the links (the leader). The network, as depicted in Figure 6, is composed of *NCustomer* customers (the followers) that route their data independently at the smallest possible cost. Customer i wishes to transmit d_i amount of data from source x_i to target y_i. Each path from a source to a target has to cross a tolled arc a_j. On the way from x_i to a_j, the cost of the links (owned by other providers) is c_{ij}. It is assumed that each customer i wishes to minimize the cost to route her data and that he can always choose another provider at a cost u_i. The purpose of the problem is to determine the cost t_j to cross a tolled arc a_j in order to maximize the revenue of the telecom operator. In Figure 6, there are 2 customers and 3 tolled arcs. This problem can be expressed as a QCOP as follows:

```
const NCustomer
const NArc
const c[NCustomer,NArc] // c[i,j] = fixed cost for Ci to reach Aj
```

```
const d[NCustomer]      // d[i] = demand for customer i
const u[NCustomer]      // u[i] = maximal price for customer i
\exists t[1], ..., t[NArc] in [0,max]
|  \forall k in [1,NCustomer]
|  |  \exists a in [1,NArc],
|  |  |     cost in [1,max],
|  |  |     income in [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)
```

We generated sets of random instances of this problem. These sets differ from each other as a consequence of different values assigned to two parameters: (1) the number of links the network operator owns, and (2) the number of clients who want to use these links. The network operator can choose between five prices for each link. For each instance, the maximum price each customer is willing to pay and the initial costs (to go from home to the starting point of a given link) are randomly chosen. Each set contains 100 instances. These tests were run on machines equipped with two dual-core opteron and 4 GB of RAM. QeCode being mono-threaded, each core was assigned one instance. No timeout has been set. The branch and bound is not activated because the condition is not met.

Figure 7 shows the average and median resolution times of these tests. Instances with a number of links smaller than 7 are not shown, as most of them were solved in less than one second. It is noteworthy that the number of clients has a very small impact on the resolution time, contrary to the number of links. This effect can be explained considering that adding clients simply requires to choose the link in which their data will transit, while adding links adds one possible choice to every client, and multiplies the pricing alternatives of the network operator.

Virtual network pricing. Telecom infrastructure requires very large investments, supported by one or a few *network operators* (NO). To increase competition,

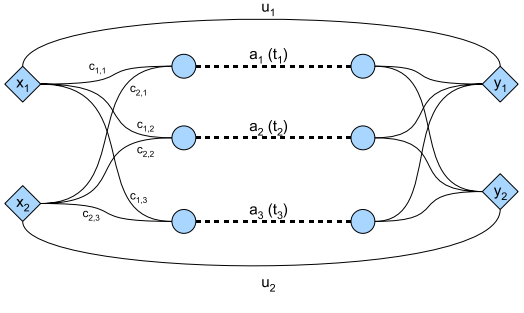

Fig. 6. A network pricing problem

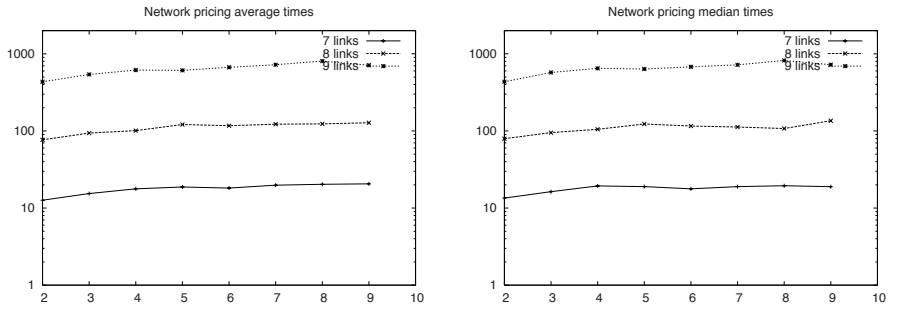

Fig. 7. Average and median resolution times (Y-axis, in seconds) on the *Network Link Pricing* problem for 7, 8 and 9 links, and for between 2 and 9 clients (X-axis)

governments have fostered the introduction of *virtual network operators* (VNO), who provide the same services except that they do not own their network. They rent capacity on the network of NO instead. To some extent, fixed by a regulating authority, network operators are simultaneously competing and cooperating. Taking good decisions in such an environment requires a model of oligopoly which is complex and far from Walras' pure and perfect competition. The following example is taken from [16].

Figure 8 depicts the relations between the NO, the VNO and the customers, each actor being modeled as in [16]. Let us assume the point of view of NO. Our main purpose is to determine the decisions $y = (y_1, y_2)$, y_1 being the service provision to our own customers and y_2 the price for capacity leased to the VNO. The decisions taken by the VNO are $z = (z_1, z_2)$, z_1 being the price for service provision to VNO's customers and z_2 the capacity leased from the NO. The customer are modeled by $n = (n_1, n_2)$ (total number of customers of NO and VNO respectively) according to the prices set for service provision: $n_i = k_i + r_{i,1}y_1 + r_{i,2}y_2$, the parameters $k_i, r_{i,1}$ and $r_{i,2}$ being determined by analysis of market data. The profit of VNO is given by the revenue of the customers minus the cost of leasing, i.e. $(q - e_2 z_1)n_2 - y_2 z_2 - g_2$, where q, e_2 and g_2 are respectively the fixed and variable costs by customer and the fixed service provision cost. The profit of the NO is given by the revenue from service provision to the customers and by the capacity allocated to the VNO, i.e. $g_1 + (q + y_1 + e_1)n_1 + y_2 z_2$ where e_1 and g_1 are respectively the variable costs by customer and the fixed service provision cost. Note that the revenue of NO depends on decisions taken by VNO. In addition, the price for service to customers is comprised between the limits d and D, the price of leasing has an upper limit of U_1 fixed by the authority, and the maximal

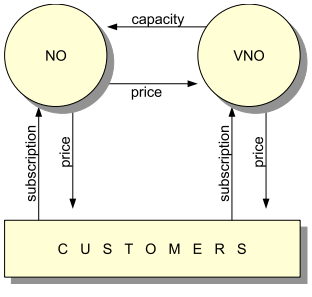

Fig. 8. Virtual network pricing

capacity available for VNO is less than a limit U_2. We construct the following QCOP$^+$ model (in which the universal quantifier is not used since there is only one virtual operator). Note that the non-overlapping condition is not met.

```
const d, D, U1, U2, k1, r11, r12, r21, r22, g1, q
\exists y1 in Dy1, y2 in Dy2
|  [d =< y1, y1 =< D, y2 =< U1]
|  \exists z1 in Dz1, z2 in Dz2
|  |  [d =< z1, z1 =< D, z2 =< U2]
|  |  \exists n1 in Dn1, n2 in Dn2, rno in Drno, rvno in 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 Related Work and Conclusion

Closely related works are the framework of *Plausibility-Feasibility-Utility* (PFU) [17], a work on *iterated expressions* [18], and the language of *Stochastic CSPs* [14]. The PFU framework aims at providing an algebraic unification of QBF, QCSP, Stochastic CSP, Bayesian networks, and Markov Decision Processes, while the purpose of iterated expressions is to model resource allocation in workflows. They both introduce expressions of the form $\bigoplus_{x_1 \in D_1} \ldots \bigoplus_{x_n \in D_n} expr(x_1, \ldots x_n)$ where $\bigoplus \in \{\min, \max, \sum, \Pi\}$. It is not possible to express bilevel models in these frameworks because the optimization condition has to apply on an expression involving the result of an immediate subexpression. However, some constructions like $\min(\sum_x e_1(x) + \sum_y e_2(y))$ are expressible by a PFU or iterated expression and not by a QCOP$^+$. Moreover, the branch and bound condition is always verified by construction. In [19] it is proposed to find a boolean QCSP strategy which maximizes the weighted sum of its existential variables set to 1. A dichotomy theorem is also proved (to identify tractable and intractable language classes).

All these works, along with QCOP$^+$, pose the problem of finding an adequate language for expressing preferences over strategies. In this respect, the QCOP$^+$ framework is general enough to model bi and multi-level problems whose importance is confirmed by several studies of the game theory and operations research.

Several unanswered questions remains: Should the objective function be compositionally defined or not? What is the analog of weighted CSP in this context? When several strategies are optimal at the first level, it can happen that they differ considerably as to sub-strategies. The language of QCOP$^+$ does not allow to capture such subtle differences. Enabling constraints on aggregate values is another issue. It is for example impossible to require that strategies should take different values of an existential variable for all values of an universal variable: It would require an "all-different" aggregate that may fail. A last open question concerns the evaluation of partial and/or non-winning strategies, which could open the way to both relaxation and local search in quantified problems.

References

1. Nightingale, P.: Consistency for quantified constraint satisfaction problems. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 792–796. Springer, Heidelberg (2005)
2. Bessière, C., Verger, G.: Strategic constraint satisfaction problems. In: Miguel, I., Prestwich, S. (eds.) Workshop on Constraint Modelling and Reformulation, Nantes, France, pp. 17–29 (2006)
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, pp. 38–43. AAAI Press, Menlo Park (2007)
4. Benedetti, M., Lallouet, A., Vautard, J.: Modeling adversary scheduling with QCSP+. In: ACM Symposium on Applied Computing, Fortaleza, Brazil. ACM Press, New York (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, 37–44 (1973)
7. Stackelberg, H.: The theory of market economy. Oxford University Press, Oxford (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, 235–256 (2007)
10. QeCode Team: QeCode: An open QCSP+ solver (2008), `http://www.univ-orleans.fr/lifo/software/qecode/`
11. Gecode Team: Gecode: Generic constraint development environment (2006), `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, pp. 47–53. Professional Book Center (2005)
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, pp. 360–365. AAAI Press, Menlo Park (2005)
14. Walsh, T.: Stochastic constraint programming. In: ECAI, pp. 111–115 (2002)
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), 411–420 (2007)
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), 19–39 (2006)
17. Pralet, C., Verfaillie, G., Schiex, T.: An algebraic graphical model for decision with uncertainties, feasibilities, and utilities. Journal of Artificial Intelligence Research 29, 421–489 (2007)
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, pp. 98–107 (2007)
19. Chen, H., Pál, M.: Optimization, games, and quantified constraint satisfaction. In: Fiala, J., Koubek, V., Kratochvíl, J. (eds.) MFCS 2004. LNCS, vol. 3153, pp. 239–250. Springer, Heidelberg (2004)