Quantified Valued Constraint Satisfaction Problem

Florent Madelaine[1(✉)] and Stéphane Secouard[2]

[1] Université Paris-Est Créteil, LACL, Créteil, France
florent.madelaine@uca.fr
[2] Université Caen Normandie, CNRS, GREYC, Caen, France

Abstract. We study the complexity of the quantified and valued extension of the constraint satisfaction problem (QVCSP) for certain classes of languages. This problem is also known as the weighted constraint satisfaction problem with min-max quantifiers [1].

The multimorphisms that preserve a language is the starting point of our analysis. We establish some situations where a QVCSP is solvable in polynomial time by formulating new algorithms or by extending the usage of collapsibility, a property well known for reducing the complexity of the quantified CSP (QCSP) from Pspace to NP. In contrast, we identify some classes of problems for which the VCSP is tractable but the QVCSP is Pspace-hard.

As a main Corollary, we derive an analogue of Shaeffer's dichotomy between P and Pspace for QCSP on Boolean languages and Cohen *et al.* dichotomy between P and NP-complete for VCSP on Boolean valued languages: we prove that the QVCSP follows a dichotomy between P and Pspace-complete.

Finally, we exhibit examples of NP-complete QVCSP for domains of size 3 and more, which suggest at best a trichotomy between P, NP-complete and Pspace-complete for the QVCSP.

Keywords: Complexity classification · Valued CSP
Quantified CSP · Polymorphisms · Multimorphisms · Collapsibility

1 Introduction

Modern SAT and CSP solvers are quite efficient on industrial instances, so much so that there is a current impetus in the community towards solvers that tackle computational problems that lie beyond NP [2]. Meanwhile on the theoretical front, several proofs [3,4] have just been proposed for Feder and Vardi celebrated dichotomy conjecture for the CSP [5]. There has been some advances for its quantified counterpart the QCSP that seems to follow a trichotomy between P, NP-complete and Pspace-complete [6–9]. Its optimisation counterpart the VCSP

Supported by Université Clermont Auvergne, CNRS, LIMOS and Université Caen normandie, CNRS, GREYC.

has been classified, first for finite valued cost functions [10] then for arbitrary cost function assuming the dichotomy conjecture holds [11]. The reader may also consult these recent surveys on QCSP [12] and VCSP [13].

It is now well established that the presence or absence of certain well behaved *polymorphisms* of a constraint language characterises the complexity of the corresponding CSP. Schaefer proved a dichotomy in the Boolean case [14], which may be reformulated as follows.

Theorem 1 [15]**.** *Let Γ be a constraint language over $D = \{0, 1\}$.*
CSP(Γ) is in P if Γ admits one of the following six good polymorphisms and NP-complete otherwise.
Good polymorphisms: $\{\text{Mjrty}, \text{Mnrty}, \text{Max}, \text{Min}, \text{Const}_0, \text{Const}_1\}$.

Here, Mjrty denotes the ternary operation that returns its repeated argument, while Mnrty is the ternary minority operation; Max and Min are binary operations that returns the maximum and minimum, respectively; Const_0 and Const_1 are the unary constant operations that sends their argument to 0 and 1 respectively. We delay further formal definitions to the next section.

For the QCSP and VCSP, *surjective polymorphisms* and *fractional polymorphisms* play the same role as polymorphisms for the CSP. As an illustration, let us state two classification results in the Boolean case. For the QCSP, the following dichotomy was announced by Schaefer [14].

Theorem 2 [15,16]**.** *Let Γ be a constraint language over $D = \{0, 1\}$.*
QCSP(Γ) is in P if Γ admits one of the following four good surjective polymorphisms and Pspace-complete otherwise.
Good surjective polymorphisms: $\{\text{Mjrty}, \text{Mnrty}, \text{Max}, \text{Min}\}$.

For the Boolean VCSP, the good multimorphisms must combine good polymorphisms from Theorem 1, as otherwise the *feasibility* of the VCSP would readily allow to solve a hard CSP.

Theorem 3 [17]**.** *Let Γ be a* valued *constraint language on $D = \{0, 1\}$.*
VCSP(Γ) is in P if it admits at least one of the following eight good multimorphisms. Otherwise, the problem is NP-hard.
Good multimorphisms for VCSP:
$\{3\text{Mjrty}, 3\text{Mnrty}, 2\text{Mjrty} + \text{Mnrty}, 2\text{Max}, 2\text{Min}, \text{Max} + \text{Min}, \text{Const}_0, \text{Const}_1\}$.

In this paper, we combine universal quantification (QCSP) and valued constraints (VCSP) and work in the framework of the Quantified Valued Constraint Satisfaction Problem (QVCSP). Unbeknownst to us until the reviewers pointed it out, this problem was in fact already introduced in [1] as the *weighted CSPs with min-max quantifiers* and studied from an experimental perspective in the context of solver designs: the authors showed that alpha-beta pruning can be adapted in this context in a relevant fashion. While their name for the problem is very natural in their context, we will stick to our terminology which makes more apparent that we merge the QCSP and the VCSP frameworks. A natural way of building a QCSP instance from a CSP instance consists in assuming that a malicious opponent or uncertainty with respect to the environment is limiting certain resources or strengthening some constraints after some decisions have been made (see for example [18] for examples of scheduling with opponents). In the same manner, we could build natural example of QVCSP from natural instances of the VCSP.

Note that some of the tractable languages for the VCSP are in fact not genuine valued languages since the costs are the same for all feasible tuples. This is the case for the multimorphisms 3Mjrty and 3Mnrty in Theorem 3 (actually this is the case for any finite domain size). For such so called *essentially crisp* languages, one can therefore deduce immediately the complexity of their QVCSP from the QCSP classification: they are both *collapsible* (so in NP) and reduce in fact to a CSP in P.

For the QCSP, the fact that the complexity drops from Pspace to NP is explained by a property known as *switchability* [19–22]. Here we will only consider a restricted form of this property known as *collapsibility*, which asserts that a language is *k-collapsible*, whenever to satisfy an input sentence, it suffices to satisfy all sentences induced from this input sentence by fixing all but a bounded number of universal variables to take the same value. This is true in particular for the languages preserved by Max or Min for $k = 1$. We will show that this approach can be applied in some cases to QVCSP as well, which will in turn lead to tractability in some cases.

In particular, we obtain a complete classification of the complexity of the QVCSP in the Boolean case.

Theorem 4 (main result). *Let Γ be a* valued *constraint language on $D = \{0, 1\}$. If Γ has one of the following good multimorphism then QVCSP(Γ) is tractable, otherwise it is Pspace-hard.*
Good multimorphisms for QVCSP:
$\{3\text{Mjrty}, 3\text{Mnrty}, 2\text{Mjrty} + \text{Mnrty}, 2\text{Max}, 2\text{Min}\}$.

The hardness part of our proof relies on a fairly non trivial case analysis of tractable languages from Theorem 3 that are not tractable according to Theorem 4. We show that we can always express in this case a cost function that is hard for the QVCSP (and was of course not hard for the VCSP). Among other, we borrow and adapt the technique of *compression* used in [17] for the proof of Theorem 3.

The paper is organised as follows. In the next section we recall definitions and notations. In Sect. 3, we introduce the QVCSP and provide some examples of Pspace-hard Boolean QVCSP. In Sect. 4, we show that some valued constraint languages are tractable for essentially trivial reasons: either because they are crisp or because they are non crisp but any instance with "too many" universal quantifiers must be rejected. In Sect. 5, we extend the notion of collapsibility from the QCSP to the valued setting of QVCSP and use it to obtain tractability results for the QVCSP. In Sect. 6, we finish the proof of Theorem 4. Finally we conclude with some remarks.

2 Preliminaries

A *VCSP instance* ϕ is a finite collection of valued constraints over some finite variable set X ranging over a label set D with value in the rationals augmented with the non feasible ∞. Here a valued constraint is given by a so-called *cost function* ρ from D^n to $\mathbb{Q} \cup \{\infty\}$ for some positive integer n (the arity) and an n-tuple of elements of X (the scope). Given the assignment $\alpha := \{\sigma_x \in D | x \in X\}$, we write $\mathrm{obj}(\phi, \alpha)$ as a short hand for $\sum_{\rho(\bar{x}) \text{ in } \phi} \rho(\alpha(\bar{x}))$ where by $\alpha(\bar{x})$ we mean that the value of each variable x from the scope $\bar{x}$ is replaced by its assigned label σ_x. The VCSP is an optimisation problem, the aim of which is to compute an α such that $\mathrm{obj}(\phi, \alpha)$ is minimum. There are two other problems that arise in the context of VCSP.

- decision: given an additional input k in $\mathbb{Q}$, is there an α such that $\mathrm{obj}(\phi, \alpha)$ is at most k?
- feasibility: is there an α such that $\mathrm{obj}(\phi, \alpha)$ is finite?

All these problems are NP-hard in general, and a standard way of better understanding the complexity is to study language restrictions, that is restricts costs functions to come from a certain set Γ. A valued constraint language is NP-hard (for the VCSP), iff it contains a finite language for which the VCSP is NP-hard. A cost function is *crisp* (resp., *essentially crisp*) if it ranges over $\{0, \infty\}$ (resp., over $\{c, \infty\}$ for some c in $\mathbb{Q}$). A language is (essentially) crisp if it contains only (essentially) crisp cost functions. The crisp language associated with a valued constraint language Γ denoted by $\mathrm{crisp}(\Gamma)$ consists of the set of corresponding relations, where $\mathrm{crisp}(\rho)(t) := 0$ iff $\rho(t) < \infty$.

An m-ary *operation* on D is a function $g : D^m \to D$. Let $\mathcal{O}_D^{(m)}$ denote the set of all m-ary operations on D. An m-ary *fractional operation* is a function ω from $\mathcal{O}_D^{(m)}$ to the positive rationals such that $\sum_g \omega(g) = 1$. The set $\{g \mid \omega(g) > 0\}$ of operations is called the *support* of ω and is denoted by $\mathrm{supp}(\omega)$.

A fractional operation ω is called an m-ary *fractional polymorphism* of a r-ary valued constraint ρ if for any tuples $t_1, t_2, \ldots, t_m$ in D^r, it holds that

$$\frac{1}{m}(\rho(t_1) + \rho(t_2) + \ldots + \rho(t_m)) \geq \sum_{g \in \mathcal{O}_D^{(m)}} \omega(g)\rho(g(t_1, t_2, \ldots, t_m)) \qquad (1)$$

where the operations g are applied component-wise. We will alternatively say that a fractional operation *improves* a cost function. A *multimorphism* is a fractional polymorphism with integral weights[1]. If ω is a fractional polymorphism of every cost function in a constraint language Γ, then ω is called a fractional polymorphism of Γ. A fractional polymorphism of a crisp language is a collection of *polymorphisms* (one identifies a crisp cost function ρ with the relation

[1] We deviate marginally from the standard definition, which would require to rescale by the arity.

$\{t|\rho(t) < \infty\}$). If ω is a fractional polymorphism of Γ, any g in supp(ω) is a polymorphism of crisp(Γ). Rewriting (1) as

$$\rho(t_1) + \rho(t_2) + \ldots + \rho(t_m) \geq \sum_{g \in \mathcal{O}_D^{(m)}} \mathbf{m}.\omega(\mathbf{g})\rho(g(t_1, t_2, \ldots, t_m)) \quad (2)$$

we will mildly abuse the notation of m-ary fractional polymorphisms in this paper, and write them as a weighted sum of operations of arity m, such that the sum of weights equals m. This explains the notation used in the statement of Theorems 3 and 4, where we listed several examples including ternary fractional operations such as 3Mjrty and 2Mjrty + Mnrty, binary ones such as 2Max and Max + Min and, unary ones $\text{Const}_0, \text{Const}_1$.

We recall some operation of importance. For a constant c in D, let Const_c denote the unary operation that always return c. Given a total order over D, let Max (resp, Min) denote the binary operation that returns the largest argument. Over the Boolean domain, we shall consider the usual order $0 < 1$. More generally, any partial order over D such that the greatest lower bound of any pair of elements exist, induces naturally a *semi-lattice operation*, which is a binary operation $\wedge$ that is idempotent ($x \wedge x = x$), associative and commutative. The element $\bot := \bigwedge_{d \in D} d$ satisfies for any x in D, $\bot \wedge x = \bot$. If there is a constant $\top$ such that for any x in D, $\top \wedge x = x$, then we say that $\top$ is a *unit* and $\wedge$ a *semi-lattice with unit.* For $1 \leq i \leq 3$, let Mjrty_i denote the ternary operation that returns the argument that occurs the most if some are equal, and its ith argument otherwise. When the domain is Boolean, we drop the unnecessary subscript. We define similarly Mnrty_i, which returns the least frequent argument if some are equal. A k-ary *Hubie operation*[2] f over D with respect to a constant c in D is a surjective operation that remains surjective even when any coordinate is fixed to c. That is, $f(c, D, \ldots, D) = f(D, c, D, \ldots, D) = \ldots = f(D, \ldots, D, c) = D$. We say that a cost function $\phi(x_1, \ldots, x_m)$ can be *expressed* by Γ if there is an instance I of VCSP(Γ) with objective function $\phi_{\mathcal{I}}(x_1, \ldots, x_m, x_{m+1}, \ldots, x_n)$, such that

$$\phi(x_1, \ldots, x_m) = \underset{x_{m+1}, \ldots, x_n}{\text{Min}} \phi_{\mathcal{I}}(x_1, \ldots, x_m, x_{m+1}, \ldots, x_n).$$

We can also easily implement cost functions by *scaling* and *translating* that is a cost function $a.\phi + b$, for any $\phi \in \Gamma$, any $a \in \mathbb{Q}^+$ and any $b \in \mathbb{Q}$. Let Γ^* be the closure of Γ under expressibility, scaling and translation. It is known that this closure preserve (in)tractability and that Γ^* is the same as the set of cost functions that are invariant under the fractional polymorphisms of Γ [13, Theorem 35].

3 Definition and Examples of QVCSP

An instance of the *quantified valued constraint satisfaction problem* (QVCSP) is defined as above with the addition of a prefix of quantification $\mathcal{P}$ applying

[2] It was anonymous in [23] and the term was coined in [21].

to all variables: that is, $\mathcal{P}$ is a strict linear order over variables where variables are either existential or universal. For convenience, we will denote the set of existential variables by $X^{\mathcal{P}}$ and universal variables by $Y^{\mathcal{P}}$. Given an existential variable x of $X^{\mathcal{P}}$, we denote by $Y_x^{\mathcal{P}}$ the set of universal variables that precede x in the prefix order given by $\mathcal{P}$. When the prefix of quantification is clear from context, we feel free to drop the superscript from our notation.

A *Skolem function* σ_x for the variable x is a D ranging function that takes as input values corresponding to the values of the universal variables that precedes it in $\mathcal{P}$, that is from $D^{|Y_x|}$ to D. If β is a family of Skolem functions for our instance $\beta = \{\sigma_x : D^{|Y_x|} \to D | x \in X\}$ (we will call such a family a *strategy* for our instance) and $\pi : Y \to D$ an assignment of the universal variables, we write $\beta \circ \pi$ for the assignment to the variables which assigns a universal variable y in Y to $\pi(y)$ and an existential variable x in X to $\sigma_x(\pi_{\upharpoonright Y_x})$, where $\pi_{\upharpoonright Y_x}$ denotes the restriction to Y_x of π.

We are now in a game setting pitching a universal player (male) and an existential player (female). Informally, she is trying to give a label to existential variables with the long term view of optimising the objective function, while he is a malicious opponent trying to prevent her from doing so. She tries to minimise the objective no matter what her opponent plays. This is reasonable if she knows that he is maliciously trying to make sure that after play her objective is as large as possible. We extend therefore the objective function to quantified valued constraints and let $\text{obj}(\phi, \beta) := \max_{\pi:Y\to D} \text{obj}(\phi, \beta \circ \pi)$. We will consider the QVCSP to be the optimisation problem, the aim of which is to compute a β such that $\text{obj}(\phi, \beta)$ is minimum. We will not as such request that β be given in full as it would be of size at least D^m where m is the number of universal variables. Instead, we will ask for a procedure that can play the underlying game according to the strategy β. Like for the VCSP, there are again two natural decision problems that arise:

– decision: given an additional input k in $\mathbb{Q}$, is there a β such that $\text{obj}(\phi, \beta)$ is at most k?
– feasibility: is there a β such that $\text{obj}(\phi, \beta)$ is finite?

Note that this definition extends naturally the usual semantic of the QCSP and the feasibility question for the QVCSP amounts to solving the QCSP for the underlying crisp language. This means that the above three problems are Pspace-hard in general, and we study in this paper their restrictions to a valued constraint language Γ.

Example 1. Let Γ_{nae} be the boolean constraint language that consists of the cost function.

$$\rho_{\text{nae}}(x, y, z) = \begin{cases} \infty & \text{if} \quad x = y = z \\ 0 & \text{otherwise} \end{cases}$$

This language is crisp and we know that the complexity for the VCSP is that of the corresponding CSP, namely NP-complete, and that for the QVCSP, we should look at the QCSP, well known to be Pspace-complete [12].

Example 2. Let Γ_{neq} be the boolean constraint language that consists of the cost function

$$\rho_{\text{neq}}(x,y) = \begin{cases} 0 & \text{if} \quad x \neq y \\ 1 & \text{otherwise} \end{cases}$$

For this non crisp language, we again get an NP-hard VCSP but not because of the feasibility which is tractable, but because of the optimisation, by reduction from MAX Sat for XOR [17]. Alternatively, one can simulate a variant of ρ_{nae}:

$$\rho'_{\text{nae}}(x,y,z) := \rho_{\text{neq}}(x,y) + \rho_{\text{neq}}(x,z) + \rho_{\text{neq}}(y,z) - 1 = \begin{cases} 2 \text{ if } x = y = z \\ 0 \text{ otherwise} \end{cases}$$

We can reduce VCSP(Γ_{nae}) to VCSP(Γ_{neq}) by replacing every occurrence of the cost function by ρ'_{nae}. The former instance holds iff the latter has a solution reaching an objective of 0. The same reduction applies for the QVCSP, whose decision version is therefore Pspace-complete.

Example 3. The following boolean cost function

$$\rho_{\text{eq}}(x,y) = \begin{cases} 0 & \text{if} \quad x = y \\ 1 & \text{otherwise} \end{cases}$$

together with two unary crisp cost functions that encodes the constants 0 and 1 forms the boolean language Γ_{cut}, whose VCSP corresponds essentially to the problem MIN-CUT and is tractable [17]. In contrast we will show below that the language $\Gamma_{\text{eq}} = \{\rho_{\text{eq}}\}$ has already a QVCSP that is Pspace-hard.

Proposition 1. *The QVCSP for the constraint language Γ_{eq} is Pspace-hard.*

Proof. We reduce the decision version of QVCSP for Γ_{neq} (see example above) to that of Γ_{eq} as follows.

Given an instance ϕ of the former with a quantifier prefix $\mathcal{P}$, we reduce to the instance $\widetilde{\phi}$ obtained by replacing every occurrence of the cost function ρ_{neq} by ρ_{eq} in ϕ and picking the dual quantifier prefix $\widetilde{\mathcal{P}}$ (that is turn existential variables to universal and *vice versa*).

Let N be the number of occurrences of the ρ_{neq} cost function in ϕ. We claim that the objective for $\widetilde{\phi}$ must be more than $N - k$, iff the objective for ϕ is less then k.

Indeed, otherwise pitting a strategy $\widetilde{\beta}$ for $\widetilde{\phi}$ that would attain an objective of less than N minus k, against any strategy β for ϕ in the game for ϕ, we would obtain a final objective of more than k.

The dual argument applies for the other direction, which proves our claim.

The claim gives us the (Turing) reduction. We answer the opposite answer of that for $\widetilde{\phi}$ with the threshold $N - k$. $\square$

4 Some Tractable Languages

4.1 Essentially Crisp Languages

We can deduce the complexity of such languages from the complexity of the associated QCSP.

Proposition 2. *Let Γ be a valued constraint language over some finite set D. If Γ admits* 3Mjrty *or* 3Mnrty *as a multimorphism, where* Mjrty *(respectively,* Mnrty*) is any majority (respectively, minority) operation, then QVCSP(Γ) is tractable.*

Proof. By Proposition 6.20 (majority) and 6.22 (minority) in [17], Γ is an essentially crisp language. Thus the problem QVCSP(Γ) is the same as QCSP(Γ') where $\Gamma' = \text{crisp}(\Gamma)$. By construction, Γ' admits Mjrty or Mnrty as a polymorphism, which are known to be tractable by Theorem 4.2 (mal'tsev) and 4.5 (near-unanimity) in [6]. □

Remark 1. The above can be generalised to a language that admits a multimorphism $3f$ where f is Mal'tsev or a multimorphism $k.f$ where f is a k-ary near-unanimity operation.

4.2 Permutations and Unary

The proof principle used to discard universal quantifiers for the language of the following result is reminiscent of the case of a language that consists of a single bipartite or a single disconnected graph for the QCSP [24]. In a nutshell, an instance boils down to a collection (conjunction) of instances with a prefix of quantification with at most one leading universal variable or it must be rejected.

Theorem 5. *Let Γ be a valued constraint language over some finite set D. If Γ admits* $\text{Mjrty}_1 + \text{Mjrty}_2 + \text{Mnrty}_3$ *as a multimorphism, then QVCSP(Γ) is tractable.*

Proof. By Theorem 6.25 in [17], any cost function from Γ can be expressed as a sum of unary cost functions and binary permutation restrictions. The latter are crisp cost functions with costs ranging in $\{0, \infty\}$, that amount to a restricted permutation in the sense that for any x, there is at most one y_2 such that $\phi(x, y_2)$ holds (has non ∞ weight) and at most one y_1 such that $\phi(y_1, x)$ holds.

Our algorithm will apply some simple preprocessing and detect that the instance is not feasible or it will deduce that each connected component of the constraint graph contains at most one universal variable and by some simple case analysis deduce the (worst) cost for these components. In effect this reduces the instance to a VCSP instance for which a simple algorithm is already known.

Let ϕ be a permutation restriction occurring in the instance.

If $\exists x \forall y \phi(x, y)$ occurs in the instance then it is not feasible since any but at most one value for y will yield an objective of ∞ for a given value of x. In this case, we may answer ∞. Of course, the symmetric case of $\exists x \forall y \phi(y, x)$ is dealt with in the same way. We ignore symmetric cases from now on.

Similarly, if $\forall y_1 \forall y_2 \phi(y_1, y_2)$ occurs in the instance, then we may answer ∞.

In the degenerate case of $\forall y \phi(y, y)$, we may also answer ∞ unless ϕ is the crisp function for equality over D, in which case, we may simply discard $\phi(y, y)$ from the sum.

So we may assume from now on that any occurrence of a permutation restriction is of the form $\forall y \exists x \phi(x, y)$. If there is one y_0 such that $|\{x|\phi(x, y_0) = 0\}| = 0$ then we may answer ∞. Otherwise, let $\zeta(y)$ be the unique x such that $\phi(x, y) = 0$. The only Skolem function for x that yields feasibility is essentially unary and depends only of $y : \sigma_x(y) = \zeta(y)$.

If there is a path $y_1, x_1, x_2, \ldots, x_n$ where y_1 is universal and $x_1, x_2, \ldots, x_n$ are existential variables in the constraint graph then there are some permutations ζ_i on D such that the only Skolem functions for these existential variables that could possibly yield feasibility are of the form $\sigma_{x_1}(y) = \zeta_1(y_1)$, $\sigma_{x_2}(y) = \zeta_2(\zeta_1(y_1)) \ldots \sigma_{x_n}(y) = \zeta_n(\ldots \zeta_2(\zeta_1(y_1)) \ldots)$. If there is an edge from x_n to some universal variable y_2 then the instance is necessarily unfeasible since any but one value for y_2 will yield an objective of ∞ for a given value of x_n.

We can solve in parallel the part of the instance induced by each connected component of the constraint graph, and we may assume that a connected component of this graph contains at most one universal variable that is quantified ahead of the existential variables of this connected component.

A connected component that does not contain any universal variable can be solved efficiently by some simple propagation (see [17]).

If a universal variable y occurs only within the scope of unary constraints then we simply assume that y takes the value yielding the worst cost.

More generally, for each connected component that contains one universal variable y, we can check in parallel for all values d of y, the corresponding cost $M_{y=d}$ for the component (all other variables are now fixed). Let M be the maximum combined cost among $M_{y=d}$. $\square$

Remark 2. The tractable languages for the QCSP from [24] mentioned above can be shown to exhibit collapsibility thanks to specific polymorphisms (see examples 2 and 3 in [23]). While we shall proceed similarly for the valued languages of the next section with a suitable multimorphism, we do not yet know of a multimorphism that witnesses directly "collapsibility" for the language of Theorem 5.

5 Collapsibility in the Valued Settings

Following Chen [20], for an input of the QVCSP with m universal variables, we restrict the universal opponent to play universal variables from a specific set of tuples, and investigate the interpolation of unrestricted game from restricted (small sized) ones in the presence of good multimorphisms.

As a concrete application, we will see the case of a language closed under the multimorphism $2.g$ (2 times g) where g is a semi-lattice with unit $\top$. On an instance involving cost functions improved by this multimorphism, we may interpolate a winning strategy from winning strategies for all instances induced by replacing all but one universal variable by $\top$.

The necessary definitions and notations to discuss such an interpolation in general can be a bit off-putting, and the keen reader may refer to the appendix. Here, we will only eventually state our tractability result and give first a detailed and concrete example to illustrate it.

Example 4. Consider the instance $\forall y_1 \exists x_1 \forall y_2 \forall y_3 \exists x_2 \ \phi(y_1, x_1, y_2, y_3, x_2)$, where ϕ is the 5-ary Boolean cost function such that the cost of $(0,0,0,0,0)$ is 51, that of $(0,0,0,0,1)$ and $(0,0,0,1,0)$ is ∞, that of $(0,0,0,1,1)$ is 21, *etc.* This instance is depicted on Fig. 1. It can be checked that the cost function admits 2Max as a multimorphism: that is, the sum of the costs of any two tuples dominates twice the cost of their max taken component-wise. For example, when $t_1 = (0,0,1,0,1)$ and $t_2 = (1,0,0,1,1)$; their max is $t_3 = (1,0,1,1,1)$; the costs are $\phi(t_1) = 51$, $\phi(t_2) = 13$ and $\phi(t_3) = 5$; it is indeed the case that $10 = 2 \times 5 \leq 51 + 13 = 64$.

We replace all but one universal variable by 0 (the value of the unit $\top$ for the specific case of Max) and derive three restricted games, which amounts to solve the instances that are depicted on Fig. 2. Of course, each restricted game is a relaxation of the original instance. Thus, if one of them is not feasible then the original instance is also not feasible. The same argument applies to the objective reached by feasible instances.

The important point is that the converse holds since we can interpolate a strategy for the original instance from three strategies for the restricted games, in a way that can only improve them.

In what follows, we will assume that we have at our disposal the three strategies that are optimal for each restricted game.

Imagine that the first universal quantifier takes value 1, that is $y_1 = 1$. We will play also $y_1 = 1$ in the first restricted game and $y_1 = 0$ in the other two restricted games (we may not do otherwise). Observe that the max of the triple $(1,0,0)$ is 1. Next, we look up where the subsequent existential variable x_1 is played in each restricted game. For example, we must have $x_1 = 1$ in the first restricted game, and $x_1 = 0$ in the two other games (otherwise, we would end up being necessarily unfeasible). We apply max to this triple $(1,0,0)$ and play $x_1 = 1$ in the original game. We proceed in this fashion going back and forth taking antecedent and image under max. For example, $y_2 = 0$ brings us back to y_2 being played on $(0,0,0)$ in the three games, and $y_3 = 1$ brings us back to y_3 being played on $(0,0,1)$ in the three games. In these three games x_2 must be played on 1,0, and 1, respectively. We play x_2 on their maximum which is 1. The "branches" of play in the four games alluded to above are highlighted on Figs. 1 and 2.

The fact that Max is surjective means that we can always go back. The fact that 2Max is a multimorphism means (with a little bit of work) that the strategy we have interpolated from those for the restricted games can only improve them.

In the previous example, we have explained how a general strategy can be interpolated from a set of strategies applying to restricted games, where we

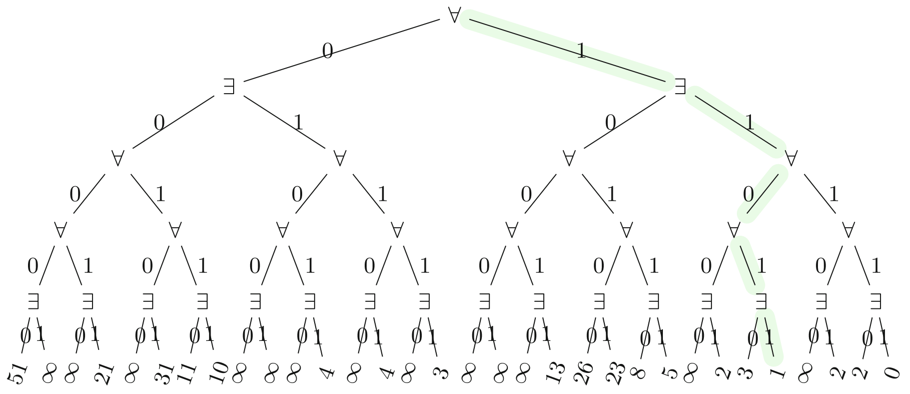

Fig. 1. An instance of the QVCSP

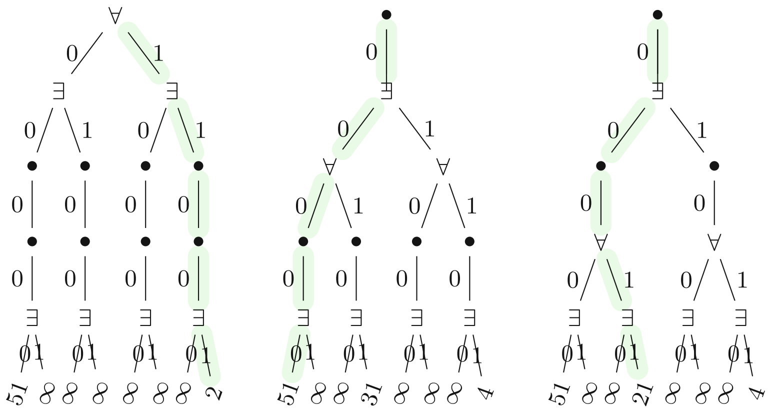

Fig. 2. Restricted Games: from left to right, we keep the first, second and third universal quantifier. The other universal variables are assumed to take value 0 (we write a • to denote that they are pinned to a constant).

are left with a single universal quantifier. Each such strategy can be computed by an adaptation of Generalized Arc Consistency that runs also in polynomial time, and there are linearly many such strategies to compute. Thus, we have a tractable QVCSP in this case.

Theorem 6. *Let Γ be a valued constraint language. Let g be a semi-lattice with unit $\top$. If Γ admits 2 times g $(2.g)$ as a multimorphism the QVCSP(Γ) is tractable.*

6 Proof of Theorem 4

We have proved in Sect. 4 that any valued Boolean constraint language that admits one of the good multimorphism from the statement is tractable: for

3Mjrty and 3Mnrty by Proposition 2, for 2Mjrty + Mnrty by Theorem 5, for 2Max and 2Min by Theorem 6. So, we are left with the hardness part of the statement.

If a constraint language does not admit any of these multimorphisms, and is essentially crisp it must be hard by Theorem 2. If a constraint language does not admit any of the multimorphisms of Theorem 3, and is not essentially crisp, then by Lemma 7.10 in [17] Γ^* contains ρ_{neq} which has a Pspace hard QVCSP as seen in Example 2.

The next lemma concludes the proof, as we show that even if a language must admit **all** multimorphisms from Theorem 3 that are not in Theorem 4, then it can simulate a cut function.

A language Γ' that admits less multimorphisms than Γ can express any finite subset of Γ^*. So any such Γ' would also simulate a cut function.

A *cut function* is a binary function from $\{0,1\}^2$ in $\mathbb{Q} \cup \{\infty\}$ of the following form $\phi_{cut_\alpha^\beta}(x,y) = \begin{cases} \alpha & \text{if } x = y, \\ \beta & \text{otherwise} \end{cases}$ where $\alpha, \beta \in \mathbb{Q} \cup \{\infty\}$ with $\alpha < \beta < \infty$. The QVCSP of a cut function is Pspace-hard by Proposition 1, since ρ_{eq} can be simulated by scaling and translating from any cut function.

Since we are in a quantified and valued setting, we will be able to use expressivity, scaling and translating as for the VCSP but also universal quantifiers as for the QCSP. We will call this *simulation with some universal quantifiers* in what follows to stress that we go beyond the $*$ closure from the VCSP.

Lemma 1. *Let Γ be a valued Boolean constraint language. If Γ admits* Const_1*,* Const_0 *and* $\mathrm{Max} + \mathrm{Min}$ *as multimorphism but no multimorphism from* $\{$3Mjrty, 3Mnrty, 2Mjrty + Mnrty, 2Max, 2Min$\}$ *then Γ can simulate with some universal quantifiers a cut function.*

Consequently, the decision problem of the QVCSP of Γ is Pspace-complete.

The rest of this section is devoted to a proof of this Lemma.

Fact 1. *If Γ admits* Max+Min *as a multimorphism then* crisp(Γ) *admits* Mjrty *as a polymorphism.*

Proof. crisp(Γ) admits both Max and Min as polymorphisms and the majority can be defined as follows:

$$\mathrm{Mjrty}(a,b,c) := \mathrm{Max}[\mathrm{Max}(\mathrm{Min}(a,b), \mathrm{Min}(a,c)), \mathrm{Min}(b,c)].$$

Fact 2. *If Γ does not admit* 3Mjrty *as a multimorphism and* crisp(Γ) *does admit* Mjrty *as a polymorphism then Γ is not essentially crisp.*

Proof. Let ρ be a cost function in Γ and u, v, w such that $\rho(u), \rho(v), \rho(w) < \infty$. Since Mjrty is a polymorphism of crisp(Γ) then $\rho(\mathrm{Mjrty}(u,v,w)) < \infty$. If ρ is essentially crisp then $3\rho(\mathrm{Mjrty}(u,v,w) = 3\rho(u) = \rho(u) + \rho(v) + \rho(w)$. If Γ was essentially crisp then it would admit 3Mjrty as a multimorphism which would contradict our assumption.

Recall that ρ is *finitely modular*, whenever for all tuples s, t such that $\phi(s)$, $\phi(t)$, $\phi(\text{Max}(s,t))$, and $\phi(\text{Min}(s,t))$ have finite costs, we have that $\phi(s) + \phi(t) = \phi(\text{Max}(s,t)) + \phi(\text{Min}(s,t))$.

Fact 3. *If Γ does not admit* $2\text{Mjrty} + \text{Mnrty}$ *as a multimorphism and* $\text{crisp}(\Gamma)$ *does admit* Mjrty *as a polymorphism then there exist a cost function in ρ that is not finitely modular or* $\text{crisp}(\rho)$ *does not admit* Mnrty *as a polymorphism.*

Proof. Corollary 6.26 in [17] establishes that a cost function ρ does admit $2\text{Mjrty} + \text{Mnrty}$ as a multimorphism iff it is both finitely modular and $\text{crisp}(\rho)$ admits as polymorphisms both Mjrty and Mnrty.

Fact 4. *If Γ is not essentially crisp, and it admits* Const_0 *and* Const_1 *as multimorphisms, and* $\text{crisp}(\Gamma)$ *admits* Mjrty *but does not admit* Mnrty *then Γ^* contains a cut function.*

Before proving this, let us point out that this means that we are only left with the case when there is a ρ that is not finitely modular, a case that we will settle in the last Fact.

Proof. We follow the same argument as in case 3 of the proof of Theorem 6.27 from [17] and establish that Γ contains a binary cost function ρ such that for exactly one $(a,b) \in D^2$ there is $\rho(a,b) = \infty$ (other values being finite). Since Γ admits Const_0 and Const_1 as multimorphisms, we know that $\rho(0,0) = \rho(1,1) \leqslant \rho(b,a) < \rho(a,b) = \infty$. W.l.o.g. up to symmetry, we can suppose that $a = 0$ and $b = 1$ and we have $\rho(0,0) = \rho(1,1) \leqslant \rho(1,0) < \rho(0,1) = \infty$.

We must ensure $\rho(1,0) > \rho(0,0)$ for our next construction to work. If it is not the case, then since Γ is not essentially crisp and has the multimorphisms Const_0 and Const_1, there is a cost function ρ_m (of arity m) and a m-tuple u such that $\rho_m(0,\ldots,0) = \rho_m(1,\ldots,1) < \rho_m(u) < \infty$. Let ρ_2 be the binary function obtained by $\rho_2(x_1,x_0) = \rho_m(x_{u[1]},\ldots,x_{u[m]})$. We do not know for sure the value of $\rho_2(0,1)$ but we know that $\rho_2(0,0) = \rho_2(1,1) < \rho_2(1,0) = \rho_m(u) < \infty$.

Let $\rho_3(x,y) := \rho(x,y) + \rho_2(x,y)$. By construction, $\rho_3(0,0) = \rho_3(1,1) < \rho_3(1,0) < \rho_3(0,1) = \infty$.

The last function which is the desired cut function is created by expressibility as follows:

$$\rho_4(x,y) := \underset{z,t}{\text{Min}}[\rho_3(x,t) + \rho_3(z,t) + \rho_3(z,y) + \rho_3(y,z) - 4\rho_3(0,0)]. \qquad \square$$

The next step will rely heavily on the technique of *compression* from [17], which we shall adapt to our purpose. Given an m-ary cost function ρ_m and two m-tuples u and v, let the *compression* ρ_4 of ρ_m w.r.t. u and v is defined as: $\rho_4(x_{00}, x_{01}, x_{10}, x_{11}) = \rho(x_{u[1]v[1]}, x_{u[2],v[2]}, \ldots, x_{u[m]v[m]})$. One can verify that $\rho_4(0,0,0,1) = \rho_m(\text{Min}(u,v))$, $\rho_4(0,0,1,1) = \rho_m(u)$, $\rho_4(0,1,0,1) = \rho_m(v)$ and $\rho_4(0,1,1,1) = \rho_m(\text{Max}(u,v))$.

Next, we want to ensure that the first and last coordinate of ρ_4 must take values 0 and 1 in order to simulate the binary cost function $\rho_4(0,x_1,x_2,1)$.

Fact 5. *If Γ is not essentially crisp and admits* Const_1 *and* Const_0 *as multimorphisms, then Γ^* contains a cut function or a small quantified instance with 2 free variables x_1 and x_2 and two universal variables built from cost functions from Γ together with ρ_4 allows to simulates the binary cost function $\rho_4(0, x_1, x_2, 1)$ for any m-ary cost function ρ_m from Γ.*

Proof. Let $\rho_2(0,0) = \rho_2(1,1) < \rho_2(1,0) = \rho_m(u) < \infty$ be defined as in the proof of the previous fact (we only need the assumptions that Γ is not essentially crisp and admits Const_1 and Const_0 as multimorphisms). If $\rho_2(0,1)$ is also finite then $\rho_2(x,y) + \rho_2(y,x)$ expresses a cut function and we are done.

Otherwise, $\exists x_1 \exists x_2 \forall y_1 \forall y_2 \quad \rho_2(x_1,y_1) + \rho_2(y_2,x_2) - 2\rho_2(1,0)$ forces $x_1 = 0$ and $x_2 = 1$ (because this is the only way to avoid an infinite cost).

So *for any cost function ρ_m* from Γ and its compression ρ_4, if we insert at the beginning of an instance $\exists x_1 \exists x_2 \forall y_1 \forall y_2 \quad \rho_2(x_1,y_1) + \rho_2(y_2,x_2) - 2\rho_2(0,1)$, all subsequent constraint of the form $\rho_4(x_1,x,y,x_2)$ plays the same role as $\rho_4(0,x_1,x_2,1)$.

Fact 6. *If there exists a cost function which is not finitely modular in Γ, which does admit* $\mathrm{Min} + \mathrm{Max}$ *as a multimorphism but does not admit either* $2\mathrm{Max}$ *or* $2\mathrm{Min}$ *as a multimorphism then Γ can simulate with some universal variables a cut function.*

Proof. Since 2Max is not a multimorphism there is a function ρ_{NMax} and u, v such that $2\rho_{\mathrm{NMax}}(Max(u,v)) > \rho_{\mathrm{NMax}}(u) + \rho_{\mathrm{NMax}}(v)$. Both $\rho_{\mathrm{NMax}}(\mathrm{Max}(u,v)) < \infty$ and $\rho_{\mathrm{NMax}}(\mathrm{Min}(u,v)) < \infty$ because $\mathrm{Min} + \mathrm{Max}$ is a multimorphism and so both Min and Max are polymorphisms of crisp(Γ).

By the binarisation method of the compression from the previous fact, either we have a cut function and we are done or we can simulate the binary function $\rho_{2\mathrm{NMax}}$ and let $\rho_M(x,y) = \rho_{2\mathrm{NMax}}(x,y) + \rho_{2\mathrm{NMax}}(y,x) - 2\rho_{2\mathrm{NMax}}(0,0)$.

$$\text{We have } \rho_M : \begin{cases} 1,1 \mapsto A + \epsilon_1 \text{ with } 0 < \epsilon_1 \leqslant A \\ 1,0 \mapsto A \text{ with } A > 0 \\ 0,1 \mapsto A \\ 0,0 \mapsto 0 \end{cases}$$

$0 < \epsilon_1$ because ρ_{NMAX} does not have the multimorphism 2Max and $\epsilon_1 \leqslant A$ because ρ_{NMAX} has the multimorphism $\mathrm{Min} + \mathrm{Max}$.

There is a function ρ_{NMod} which is not finitely modular and admits $\mathrm{Min} + \mathrm{Max}$ as a multimorphism. So there are u, v such that $\rho_{\mathrm{NMod}}(Max(u,v)) + \rho_{\mathrm{NMod}}(\mathrm{Min}(u,v)) < \rho_{\mathrm{NMod}}(u) + \rho_{\mathrm{NMod}}(v) < \infty$. By the binarisation method from the previous fact, either we have a cut function and we are done or we can simulate $\rho_{2\mathrm{NMod}}$ and define $\rho_s(x,y) := \rho_{2\mathrm{NMod}}(x,y) + \rho_{2\mathrm{NMod}}(y,x) - 2\rho_{2\mathrm{NMod}}(0,0)$.

$$\text{By construction, we have, } \rho_s : \begin{cases} 1,1 \mapsto b_0 < 2b \\ 1,0 \mapsto b \\ 0,1 \mapsto b \\ 0,0 \mapsto 0 \end{cases} .$$

Let α be a positive integer such that $\alpha > \frac{b-b_0}{\epsilon_1}$ and $\varphi_M := \alpha\rho_M + \rho_s$

- $\varphi_M(1,1) = \alpha A + \alpha\epsilon_1 + b_0 > \alpha A + b - b_0 + b_0 = \varphi_M(1,0)$
- $\varphi_M(1,1) = (A + \epsilon_1)\alpha + b_0 < 2A\alpha + 2b = 2(\alpha A + b) = 2\varphi_M(1,0)$
- $\varphi_M(1,0) = \varphi_M(0,1) = b + \alpha A \geqslant b + \alpha\epsilon_1 > b + b - b_0 > 0$
- $\varphi_M(0,0) = 0$

We have $\varphi_M : \begin{cases} 1,1 \mapsto M + \epsilon_M \text{ with } 0 < \epsilon_M < M \\ 1,0 \mapsto M \text{ with } M > 0 \\ 0,1 \mapsto M \\ 0,0 \mapsto 0 \end{cases}$

A similar proof with 2Min instead of 2Max can be used to construct the binary function ρ_m such that, $\varphi_m : \begin{cases} 1,1 \mapsto 0 \\ 1,0 \mapsto m \text{ with } m > 0 \\ 0,1 \mapsto m \\ 0,0 \mapsto m + \epsilon_m \text{ with } 0 < \epsilon_m < m \end{cases}$

We define the function $\rho(x,y) = (m + \epsilon_m)\rho_M + (M + \epsilon_M)\rho_m$ and we have:

- $\rho(1,1) = \rho(0,0) = mM + m\epsilon_M + M\epsilon_m + \epsilon_M\epsilon_m$
- $\rho(1,0) = \rho(0,1) = mM + m\epsilon_M + M\epsilon_m + mM$
- $\epsilon_M\epsilon_m < mM$

So ρ is a cut function as required. □

7 Conclusion

We have studied the quantified valued constraint satisfaction problem, also known as the weighted CSPs with min-max quantifiers, and established preliminary results regarding its complexity when restricted by a valued language.

Without introducing any new Galois connection and using only the tools for the VCSP, and only adapting collapsibility from the QCSP, we get several tractability and intractability results, which allows us to derive in particular a dichotomy for the Boolean case. The proof is somewhat complex, and we plan to introduce the correct Galois connection for the QVCSP in the hope that it will allow to streamline this proof, and extend this result to larger domains.

Another line of enquiry would be to better understand collapsibility in the context of valued constraints. Our current attempt does not seem to provide us with transitivity as it does in the non valued case.

Finally, let us note that any attempt at classifying the QVCSP for 3 or more elements might hit the same hurdle as in the case of the QCSP. We can easily build problems that fall in NPO and are NP-hard. For example consider

$$\rho : \begin{cases} \{0,1,2\} \to \mathbb{Q} \cup \{\infty\} \\ (x,y) \mapsto \begin{cases} \rho_{\text{neq}}(x,y) \text{ if } (x,y) \in \{0,1\} \\ \infty \text{ otherwise} \end{cases} \end{cases}$$

Every instance with a universal quantifier y can be trivially answered as the objective is ∞ as soon as y is 2. We are left with existential instances which likewise must play on $\{0, 1\}$. Consequently, QVCSP has the same complexity as the VCSP on ρ_{neq}, namely it is NP-hard and in NPO.

Acknowledgments. The authors are thankful to the three anonymous reviewers for their valuable comments which have helped us improve the manuscript.

References

1. Lee, J.H., Mak, T.W.K., Yip, J.: Weighted constraint satisfaction problems with min-max quantifiers. In: IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, Boca Raton, FL, USA, 7–9 November 2011, pp. 769–776. IEEE Computer Society (2011). https://doi.org/10.1109/ICTAI.2011.121
2. Beyond np. http://beyondnp.org/. Accessed 21 June 2017
3. Bulatov, A.A.: A dichotomy theorem for nonuniform CSPs. In: Umans [26], pp. 319–330. https://doi.org/10.1109/FOCS.2017.37
4. Zhuk, D.: A proof of CSP dichotomy conjecture. In: Umans [26], pp. 331–342. https://doi.org/10.1109/FOCS.2017.38
5. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. SIAM J. Comput. **28**(1), 57–104 (1998). https://doi.org/10.1137/S0097539794266766
6. Börner, F., Bulatov, A.A., Chen, H., Jeavons, P., Krokhin, A.A.: The complexity of constraint satisfaction games and QCSP. Inf. Comput. **207**(9), 923–944 (2009)
7. Martin, B.: QCSP on partially reflexive forests. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 546–560. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23786-7_42
8. Madelaine, F., Martin, B.: QCSP on partially reflexive cycles – the wavy line of tractability. In: Bulatov, A.A., Shur, A.M. (eds.) CSR 2013. LNCS, vol. 7913, pp. 322–333. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38536-0_28
9. Dapic, P., Markovic, P., Martin, B.: Quantified constraint satisfaction problem on semicomplete digraphs. ACM Trans. Comput. Log. **18**(1), 2:1–2:47 (2017). https://doi.org/10.1145/3007899
10. Thapper, J., Zivny, S.: The complexity of finite-valued CSPs. J. ACM **63**(4), 37:1–37:33 (2016). https://doi.org/10.1145/2974019
11. Kolmogorov, V., Krokhin, A.A., Rolinek, M.: The complexity of general-valued CSPs. In: Guruswami, V. (ed.) IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17–20 October 2015, pp. 1246–1258. IEEE Computer Society (2015). https://doi.org/10.1109/FOCS.2015.80
12. Martin, B.: Quantified constraints in twenty seventeen. In: Krokhin and Zivny [25], pp. 327–346. https://doi.org/10.4230/DFU.Vol7.15301.12
13. Krokhin, A.A., Zivny, S.: The complexity of valued CSPs. In: The Constraint Satisfaction Problem: Complexity and Approximability [25], pp. 233–266. https://doi.org/10.4230/DFU.Vol7.15301.9
14. Schaefer, T.: The complexity of satisfiability problems. In: STOC (1978)
15. Creignou, N., Khanna, S., Sudan, M.: Complexity Classifications of Boolean Constraint Satisfaction Problems. Society for Industrial and Applied Mathematics, Philadelphia (2001)

16. Dalmau, V.: Some dichotomy theorems on constant-free quantified boolean formulas. Technical report LSI-97-43-R., Departament LSI, Universitat Pompeu Fabra (1997)
17. Cohen, D.A., Cooper, M.C., Jeavons, P., Krokhin, A.A.: The complexity of soft constraint satisfaction. Artif. Intell. **170**(11), 983–1016 (2006). https://doi.org/10.1016/j.artint.2006.04.002
18. Benedetti, M., Lallouet, A., Vautard, J.: Modeling adversary scheduling with qcsp$^+$. In: Wainwright, R.L., Haddad, H. (eds.) Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, 16–20 March 2008, pp. 151–155. ACM (2008). https://doi.org/10.1145/1363686.1363727
19. Chen, H.: The complexity of quantified constraint satisfaction: collapsibility, sink algebras, and the three-element case. SIAM J. Comput. **37**(5), 1674–1701 (2008)
20. Chen, H.: Quantified constraint satisfaction and the polynomially generated powers property. Algebra Universalis **65**(3), 213–241 (2011). https://doi.org/10.1007/s00012-011-0125-4. an extended abstract appeared in ICALP B 2008
21. Carvalho, C., Madelaine, F.R., Martin, B.: From complexity to algebra and back: digraph classes, collapsibility, and the PGP. In: 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, 6–10 July 2015, pp. 462–474. IEEE Computer Society (2015). https://doi.org/10.1109/LICS.2015.50
22. Carvalho, C., Martin, B., Zhuk, D.: The complexity of quantified constraints. CoRR abs/1701.04086 (2017). http://arxiv.org/abs/1701.04086
23. Chen, H.: Meditations on quantified constraint satisfaction. CoRR abs/1201.6306 (2012)
24. Martin, B., Madelaine, F.: Towards a trichotomy for quantified H-Coloring. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 342–352. Springer, Heidelberg (2006). https://doi.org/10.1007/11780342_36
25. Krokhin, A.A., Zivny, S. (eds.): The Constraint Satisfaction Problem: Complexity and Approximability, Dagstuhl Follow-Ups, vol. 7. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017). http://www.dagstuhl.de/dagpub/978-3-95977-003-3
26. Umans, C. (ed.): 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, 15–17 October 2017. IEEE Computer Society (2017). http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8100284