

The Expressive Power of Valued Constraints: Hierarchies and Collapses

David A. Cohen¹, Peter G. Jeavons², and Stanislav Živný²

¹ Department of Computer Science, Royal Holloway, University of London, UK
d.cohen@rhul.ac.uk

² Computing Laboratory, University of Oxford, UK
{peter.jeavons,stanislav.zivny}@comlab.ox.ac.uk

Abstract. In this paper we investigate the ways in which a fixed collection of valued constraints can be combined to express other valued constraints. We show that in some cases a large class of valued constraints, of all possible arities, can be expressed by using valued constraints of a fixed finite arity. We also show that some simple classes of valued constraints, including the set of all monotonic valued constraints with finite cost values, cannot be expressed by a subset of any fixed finite arity, and hence form an infinite hierarchy.

1 Introduction

Building a computational model of a combinatorial problem means capturing the requirements and optimisation criteria of the problem using the resources available in some given computational system. Modelling such problems using *constraints* means expressing the requirements and optimisation criteria using some combination of basic constraints provided by the system. In this paper we investigate what kinds of relationships and functions can be expressed using a given set of allowed constraint types.

The classical constraint satisfaction problem (CSP) model considers only the feasibility of satisfying a collection of simultaneous requirements. Various extensions have been proposed to this model to allow it to deal with different kinds of optimisation criteria or preferences between different feasible solutions. Two very general extended frameworks that have been proposed are the semi-ring CSP framework and the valued CSP (VCSP) framework [2]. The semi-ring framework is slightly more general, but the VCSP framework is simpler, and sufficiently powerful to describe many important classes of problems [6,18].

In this paper we work with the VCSP framework. In this framework every constraint has an associated cost function which assigns a cost to every tuple of values for the variables in the scope of the constraint. The set of cost functions used in the description of the problem is called the *valued constraint language*.

As with all computing paradigms, it is desirable for many purposes to have a small language which can be used to describe a large collection of problems. Determining which problems can be expressed in a given language is therefore

a central issue in assessing the flexibility and usefulness of a constraint system, and it is this question that we investigate here.

We make use of a number of algebraic tools that have been developed for this question [15], and for the related question of determining the complexity of a constraint language [3,6]. By applying these tools to particular valued constraint languages, we show that some simple constraint classes provide infinite hierarchies of greater and greater expressive power, whereas other classes collapse to sets of cost functions of fixed arity which can express all the other cost functions in the class.

The paper is organised as follows. In Section 2, we define the standard valued constraint satisfaction problem and the notion of expressibility for valued constraints. In Section 3, we describe some algebraic techniques that have been developed for valued constraints in earlier papers and show how they can be used to investigate expressibility. In Section 4, we present our results. We show that some valued constraints of fixed arities can express constraints of all possible arities whereas some other sets of valued constraints cannot be expressed by any subset of fixed finite arity. Due to the page limit we only state our results, but all proofs are given in the full version of this paper [7]. Finally in Section 5, we summarise our results and suggest some important open questions.

2 Valued Constraints and Expressibility

In this section we define the valued constraint satisfaction problem and discuss how the cost functions used to define valued constraints can be combined to express other valued constraints. More detailed discussion of the valued constraint framework, and illustrative examples, can be found in [2,6].

Definition 1. A *valuation structure*, Ω , is a totally ordered set, with a minimum and a maximum element (denoted 0 and ∞), together with a commutative, associative binary **aggregation operator**, \oplus , such that for all $\alpha, \beta, \gamma \in \Omega$, $\alpha \oplus 0 = \alpha$ and $\alpha \oplus \gamma \geq \beta \oplus \gamma$ whenever $\alpha \geq \beta$.

Definition 2. An instance of the **valued constraint satisfaction problem**, VCSP, is a tuple $\mathcal{P} = \langle V, D, \mathcal{C}, \Omega \rangle$ where:

- V is a finite set of **variables**;
- D is a finite set of possible **values**;
- Ω is a valuation structure representing possible **costs**;
- \mathcal{C} is a set of **valued constraints**. Each element of \mathcal{C} is a pair $c = \langle \sigma, \phi \rangle$ where σ is a tuple of variables called the **scope** of c , and $\phi \in \Gamma$ is a mapping from $D^{|\sigma|}$ to Ω , called the **cost function** of c .

Definition 3. For any VCSP instance $\mathcal{P} = \langle V, D, \mathcal{C}, \Omega \rangle$, an **assignment** for \mathcal{P} is a mapping $s : V \rightarrow D$. The **cost** of an assignment s , denoted $Cost_{\mathcal{P}}(s)$, is given by the aggregation of the costs for the restrictions of s onto each constraint scope, that is,

$$Cost_{\mathcal{P}}(s) \stackrel{\text{def}}{=} \bigoplus_{\langle \langle v_1, v_2, \dots, v_m \rangle, \phi \rangle \in \mathcal{C}} \phi(\langle s(v_1), s(v_2), \dots, s(v_m) \rangle).$$

A **solution** to \mathcal{P} is an assignment with minimal cost.

The complexity of finding an optimal solution to a valued constraint problem will obviously depend on the forms of valued constraints which are allowed in the problem [6]. In order to investigate different families of valued constraint problems with different sets of allowed constraint types, we use the notion of a **valued constraint language**, which is simply a set of possible cost functions mapping D^k to Ω , for some fixed set D and some fixed valuation structure Ω . The class of all VCSP instances where the cost functions of the valued constraints are all contained in a valued constraint language Γ will be denoted $\text{VCSP}(\Gamma)$.

In any VCSP instance, the variables listed in the scope of each valued constraint are explicitly constrained, in the sense that each possible combination of values for those variables is associated with a given cost. Moreover, if we choose *any* subset of the variables, then their values are constrained implicitly in the same way, due to the combined effect of the valued constraints. This motivates the concept of **expressibility** for cost functions, which is defined as follows:

Definition 4. For any VCSP instance $\mathcal{P} = \langle V, D, \mathcal{C}, \Omega \rangle$, and any list $l = \langle v_1, \dots, v_m \rangle$ of variables of \mathcal{P} , the **projection** of \mathcal{P} onto l , denoted $\pi_l(\mathcal{P})$, is the m -ary cost function defined as follows:

$$\pi_l(\mathcal{P})(x_1, \dots, x_m) \stackrel{\text{def}}{=} \min_{\{s: V \rightarrow D \mid \langle s(v_1), \dots, s(v_m) \rangle = \langle x_1, \dots, x_m \rangle\}} \text{Cost}_{\mathcal{P}}(s).$$

We say that a cost function ϕ is **expressible** over a valued constraint language Γ if there exists an instance $\mathcal{P} \in \text{VCSP}(\Gamma)$ and a list l of variables of \mathcal{P} such that $\pi_l(\mathcal{P}) = \phi$. We call the pair $\langle \mathcal{P}, l \rangle$ a **gadget** for expressing ϕ over Γ .

Note that any cost function expressible over Γ can be added to Γ without changing the complexity of $\text{VCSP}(\Gamma)$.

In this paper we shall examine the expressibility of cost functions over three particular valuation structures which can be used to model a wide variety of problems [6]:

Definition 5. Let Ω be a valuation structure and let $\phi : D^m \rightarrow \Omega$ be a cost function.

- If $\Omega = \{0, \infty\}$, then we call ϕ a **crisp** cost function.
- If $\Omega = \mathbb{Q}_+$, the set of non-negative rational numbers with the standard addition operation, $+$, then we call ϕ a **finite-valued** cost function.
- If $\Omega = \overline{\mathbb{Q}}_+$, the set of non-negative rational numbers together with infinity, with the standard addition operation (extended so that $a + \infty = \infty$, for every $a \in \overline{\mathbb{Q}}_+$), then we call ϕ a **general** cost function.

Note that with any *relation* R over D we can associate a crisp cost function ϕ_R on D which maps tuples in R to 0 and tuples not in R to ∞ . On the other hand, with any m -ary cost function ϕ we can associate an m -ary crisp cost function defined by:

$$\text{Feas}(\phi)(x_1, \dots, x_m) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } \phi(x_1, \dots, x_m) = \infty \\ 0 & \text{if } \phi(x_1, \dots, x_m) < \infty. \end{cases}$$

3 Expressive Power and Algebraic Properties

Adding a finite constant to any cost function does not alter the relative costs. Hence, for any valued constraint language Γ with costs in Ω , we define the **expressive power** of Γ , denoted $\langle \Gamma \rangle$, to be the set of all cost functions ϕ such that $\phi + c$ is expressible over Γ for some constant $c \in \Omega$ where $c < \infty$.

A number of algebraic techniques to determine the expressive power of a given valued constraint language have been developed in earlier papers. To make use of these techniques, we first need to define some key terms.

The i -th component of a tuple t will be denoted by $t[i]$. Note that any operation on a set D can be extended to tuples over D in the following way. For any function $f : D^k \rightarrow D$, and any collection of tuples $t_1, \dots, t_k \in D^m$, define $f(t_1, \dots, t_k) \in D^m$ to be the tuple $\langle f(t_1[1], \dots, t_k[1]), \dots, f(t_1[m], \dots, t_k[m]) \rangle$.

Definition 6 ([9]). *Let R be an m -ary relation over a finite set D and let f be a k -ary operation on D . Then f is a **polymorphism** of R if $f(t_1, \dots, t_k) \in R$ for all choices of $t_1, \dots, t_k \in R$.*

A valued constraint language, Γ , which contains only crisp cost functions (= relations) will be called a crisp constraint language. We will say that f is a polymorphism of a crisp constraint language Γ if f is a polymorphism of every relation in Γ . The set of all polymorphisms of Γ will be denoted $\text{Pol}(\Gamma)$.

It follows from the results of [13] that the expressive power of a crisp constraint language is fully characterised by its polymorphisms:

Theorem 7 ([13]). *For any crisp constraint language Γ over a finite set*

$$R \in \langle \Gamma \rangle \Leftrightarrow \text{Pol}(\Gamma) \subseteq \text{Pol}(\{R\}).$$

Hence, a crisp cost function ϕ is expressible over a crisp constraint language Γ if and only if it has all the polymorphisms of Γ .

We can extend the idea of polymorphisms to arbitrary valued constraint languages by considering the corresponding feasibility relations:

Definition 8 ([3]). *The **feasibility polymorphisms** of a valued constraint language Γ are the polymorphisms of the corresponding crisp feasibility cost functions, that is,*

$$\text{FPol}(\Gamma) \stackrel{\text{def}}{=} \text{Pol}(\{\text{Feas}(\phi) \mid \phi \in \Gamma\}).$$

However, to fully capture the expressive power of valued constraint languages it is necessary to consider more general algebraic properties, such as the following:

Definition 9 ([4]). *A list of functions, $\langle f_1, \dots, f_k \rangle$, where each f_i is a function from D^k to D , is called a k -ary **multimorphism** of a cost function $\phi : D^m \rightarrow \Omega$ if, for all $t_1, \dots, t_k \in D^m$, we have*

$$\sum_{i=1}^k \phi(t_i) \geq \sum_{i=1}^k \phi(f_i(t_1, \dots, t_k)).$$

The next result shows that the multimorphisms of a valued constraint language are preserved by all the cost functions expressible over that language.

Theorem 10 ([6]). *If \mathcal{F} is a multimorphism of a valued constraint language Γ , then \mathcal{F} is a multimorphism of $\langle \Gamma \rangle$.*

Hence, to show that a cost function ϕ is *not* expressible over a valued constraint language Γ it is sufficient to identify some multimorphism of Γ which is not a multimorphism of ϕ .

It is currently an open question whether the set of multimorphisms of a valued constraint language completely characterizes the expressive power of that language. However, it was shown in [3] that the expressive power of a valued constraint language can be characterised by generalising the notion of multimorphism a little, to a property called a *fractional polymorphism*, which is essentially a multimorphism where each component function has an associated weight value.

Definition 11 ([3]). *A k -ary **weighted function** \mathcal{F} on a set D is a set of the form $\{\langle r_1, f_1 \rangle, \dots, \langle r_n, f_n \rangle\}$ where each r_i is a non-negative rational number such that $\sum_{i=1}^n r_i = k$ and each f_i is a distinct function from D^k to D .*

*For any m -ary cost function ϕ , we say that a k -ary weighted function \mathcal{F} is a k -ary **fractional polymorphism** of ϕ if, for all $t_1, \dots, t_k \in D^m$,*

$$\sum_{i=1}^k \phi(t_i) \geq \sum_{i=1}^n r_i \phi(f_i(t_1, \dots, t_k)).$$

For any valued constraint language Γ , we will say that \mathcal{F} is a fractional polymorphism of Γ if \mathcal{F} is a fractional polymorphism of every cost function in Γ . The set of all fractional polymorphisms of Γ will be denoted $\text{fPol}(\Gamma)$.

It was shown in [3] that the feasibility polymorphisms and fractional polymorphisms of a valued constraint language effectively determine its expressive power.

Theorem 12 ([3]).

Let Γ be a valued constraint language with costs in $\overline{\mathbb{Q}}_+$ such that, for all $\phi \in \Gamma$, and all $c \in \mathbb{Q}_+$, $c\phi \in \Gamma$ and $\text{Feas}(\phi) \in \Gamma$.

$$\phi \in \langle \Gamma \rangle \Leftrightarrow \text{FPol}(\Gamma) \subseteq \text{FPol}(\{\phi\}) \wedge \text{fPol}(\Gamma) \subseteq \text{fPol}(\{\phi\}).$$

4 Results

In this section we present our results. We consider the expressive power of crisp, finite-valued and general constraint languages. We consider the languages containing all cost functions up to some fixed arity over some fixed domain, and we also consider an important subset of these cost functions defined for totally ordered domains, the so-called *max-closed* relations, which are defined below.

The function MAX denotes the standard binary function which returns the larger of its two arguments.

Definition 13. A cost function ϕ is **max-closed** if $\langle \text{MAX}, \text{MAX} \rangle \in \text{Mul}(\{\phi\})$.

Definition 14. For every $d \geq 2$ we define the following:

- $\mathbf{R}_{d,m}$ ($\mathbf{F}_{d,m}$, $\mathbf{G}_{d,m}$ respectively) denotes the set of all crisp (finite-valued, general respectively) cost functions over a domain of size d of arity at most m , and $\mathbf{R}_d = \cup_{m \geq 0} \mathbf{R}_{d,m}$, $\mathbf{F}_d = \cup_{m \geq 0} \mathbf{F}_{d,m}$, and $\mathbf{G}_d = \cup_{m \geq 0} \mathbf{G}_{d,m}$;
- $\mathbf{R}_{d,m}^{\max}$ ($\mathbf{F}_{d,m}^{\max}$, $\mathbf{G}_{d,m}^{\max}$ respectively) denotes the set of all crisp (finite-valued, general respectively) max-closed cost functions over an ordered domain of size d of arity at most m , and $\mathbf{R}_d^{\max} = \cup_{m \geq 0} \mathbf{R}_{d,m}^{\max}$, $\mathbf{F}_d^{\max} = \cup_{m \geq 0} \mathbf{F}_{d,m}^{\max}$, and $\mathbf{G}_d^{\max} = \cup_{m \geq 0} \mathbf{G}_{d,m}^{\max}$.

Theorem 15. For all $f \geq 2$ and $d \geq 3$,

1. $\langle \mathbf{R}_{2,1} \rangle \subsetneq \langle \mathbf{R}_{2,2} \rangle \subsetneq \langle \mathbf{R}_{2,3} \rangle = \mathbf{R}_2$; $\langle \mathbf{R}_{d,1} \rangle \subsetneq \langle \mathbf{R}_{d,2} \rangle = \mathbf{R}_d$.
2. $\langle \mathbf{R}_{2,1}^{\max} \rangle \subsetneq \langle \mathbf{R}_{2,2}^{\max} \rangle \subsetneq \langle \mathbf{R}_{2,3}^{\max} \rangle = \mathbf{R}_2^{\max}$; $\langle \mathbf{R}_{d,1}^{\max} \rangle \subsetneq \langle \mathbf{R}_{d,2}^{\max} \rangle = \mathbf{R}_d^{\max}$.
3. $\langle \mathbf{F}_{f,1}^{\max} \rangle \subsetneq \langle \mathbf{F}_{f,2}^{\max} \rangle \subsetneq \langle \mathbf{F}_{f,3}^{\max} \rangle \subsetneq \langle \mathbf{F}_{f,4}^{\max} \rangle \dots$
4. $\langle \mathbf{G}_{2,1}^{\max} \rangle \subsetneq \langle \mathbf{G}_{2,2}^{\max} \rangle \subsetneq \langle \mathbf{G}_{2,3}^{\max} \rangle = \mathbf{G}_2^{\max}$; $\langle \mathbf{G}_{d,1}^{\max} \rangle \subsetneq \langle \mathbf{G}_{d,2}^{\max} \rangle = \mathbf{G}_d^{\max}$.

The proof of Theorem 15 can be found in the full version of this paper [7], and we just give a brief sketch here. As any relation can be expressed as a propositional formula, the collapse described in Statement (1) follows from the standard SAT to 3-SAT reduction. The collapse of max-closed relations in Statement (2) is proved by adapting the SAT to 3-SAT reduction. However, this gives only a weaker result, $\mathbf{R}_d^{\max} = \langle \mathbf{R}_{d,3}^{\max} \rangle$. To show that any max-closed relation over a non-Boolean domain can be expressed by using only binary max-closed relations, we characterise the polymorphisms of \mathbf{R}_d^{\max} and prove that $\mathbf{R}_{d,2}^{\max}$ does not have any extra polymorphisms. The separation result in Statement (3) is obtained by finding explicit multimorphisms for the finite-valued max-closed cost functions of each different arity. Finally, the collapse result in Statement (4) follows from a precise characterisation of the feasibility polymorphisms and fractional polymorphisms of \mathbf{G}_d^{\max} obtained using the MIN-CUT MAX-FLOW theorem.

5 Conclusions and Open Problems

We have investigated the expressive power of valued constraints in general and max-closed valued constraints in particular.

In the case of relations, we built on previously known results about the expressibility of an arbitrary relation in terms of binary or ternary relations. We were able to prove in a similar way that an arbitrary max-closed relation can be expressed using binary or ternary max-closed relations. The results about the collapse of the set of all relations and all max-closed relations contrast sharply with the case of finite-valued max-closed cost functions, where we showed an infinite hierarchy. This shows that the VCSP is not just a minor generalisation of the CSP – finite-valued max-closed cost functions behave very differently from crisp max-closed cost functions with respect to expressive power. Finally, we showed the collapse of general max-closed cost functions by investigating their feasibility

polymorphisms and fractional polymorphisms. This shows that allowing infinite costs in max-closed cost functions increases their expressive power substantially, and in fact allows them to express more finite-valued cost functions.

We remark that all of our results about max-closed cost functions obviously have equivalent versions for *min-closed* cost functions, that is, those which have the multimorphism $\langle \text{MIN}, \text{MIN} \rangle$. In the Boolean crisp case these are precisely the relations that can be expressed by a conjunction of **Horn** clauses.

One of the reasons why understanding the expressive power of valued constraints is important is for the investigation of **submodular functions**. A cost function ϕ is called submodular if it has the multimorphism $\langle \text{MIN}, \text{MAX} \rangle$. The standard problem of submodular function minimisation corresponds to solving a VCSP with submodular cost functions over the Boolean domain [5].

Submodular function minimisation (SFM) is a central problem in discrete optimisation, with links to many different areas [10,16]. Although it has been known for a long time that the ellipsoid algorithm can be used to solve SFM in polynomial time, this algorithm is not efficient in practice. Relatively recently, several new strongly polynomial combinatorial algorithms have been discovered for SFM [10,11,12]. Unfortunately, the time complexity of the fastest published algorithm for SFM is roughly of an order of $O(n^7)$ where n is the total number of variables [11].

However, for certain special cases of SFM, more efficient algorithms are known to exist. For example, the (weighted) MIN-CUT problem is a special case of SFM that can be solved in cubic time [10]. Moreover, it is known that SFM over a Boolean domain can be solved in $O(n^3)$ time when the submodular function f satisfies various extra conditions [1,8,17]. In particular, in the case of non-Boolean domains, a cubic-time algorithm exists for SFM when f can be expressed as a sum of *binary* submodular functions [5].

These observations naturally raise the following question: What is the most general class of submodular functions that can be minimised in cubic time (or better)? One way to tackle this question is to investigate the expressive power of particular submodular functions which are known to be solvable in cubic time. Any fixed set of functions which can be *expressed* using such functions will have the same complexity [3].

One intriguing result is already known for submodular *relations*. In the case of relations, having $\langle \text{MIN}, \text{MAX} \rangle$ as a multimorphism implies having both MIN and MAX as polymorphisms. The ternary MEDIAN operation can be obtained by composing the operations MAX and MIN, so all submodular relations have the MEDIAN operation as a polymorphism. It follows that submodular relations are binary decomposable [14], and hence all submodular relations are expressible using binary submodular relations.

For finite-valued and general submodular cost functions it is an important open question whether they can be expressed using submodular cost functions of some fixed arity. If they can, then this raises the possibility of designing new, more efficient, algorithms for submodular function minimisation.

Acknowledgements. The authors would like to thank Martin Cooper, Martin Green, Chris Jefferson and András Salamon for many helpful discussions.

References

1. Billionet, A., Minoux, M.: Maximizing a supermodular pseudo-boolean function: a polynomial algorithm for cubic functions. *Discrete Applied Mathematics* **12** (1985) 1–11
2. Bistarelli, S., Fargier, H., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G.: Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints* **4** (1999) 199–240
3. Cohen, D., Cooper, M., Jeavons, P.: An algebraic characterisation of complexity for valued constraints. In: *Proceedings CP'06*. Volume 4204 of *Lecture Notes in Computer Science*, Springer-Verlag (2006) 107–121
4. Cohen, D., Cooper, M., Jeavons, P., Krokhin, A.: Soft constraints: Complexity and multimorphisms. In: *Proceedings CP'03*. Volume 2833 of *Lecture Notes in Computer Science*, Springer-Verlag (2003) 244–258
5. Cohen, D., Cooper, M., Jeavons, P., Krokhin, A.: A maximal tractable class of soft constraints. *Journal of Artificial Intelligence Research (JAIR)* **22** (2004) 1–22
6. Cohen, D., Cooper, M., Jeavons, P., Krokhin, A.: The complexity of soft constraint satisfaction. *Artificial Intelligence* **170** (2006) 983–1016
7. Cohen, D., Jeavons, P., Živný, S.: The expressive power of valued constraints: Hierarchies and Collapses. Technical Report CS-RR-07-02, Computing Laboratory, University of Oxford, Oxford, UK (April 2007)
8. Creignou, N., Khanna, S., Sudan, M.: Complexity Classification of Boolean Constraint Satisfaction Problems. Volume 7 of *SIAM Monographs on Discrete Mathematics and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA. (2001)
9. Denecke, K., Wismath, S.: *Universal Algebra and Applications in Theoretical Computer Science*. Chapman and Hall/CRC Press (2002)
10. Fujishige, S.: *Submodular Functions and Optimization*. 2nd edn. Volume 58 of *Annals of Discrete Mathematics*. Elsevier (2005)
11. Iwata, S.: A faster scaling algorithm for minimizing submodular functions. *SIAM Journal on Computing* **32** (2003) 833–840
12. Iwata, S., Fleischer, L., Fujishige, S.: A combinatorial, strongly polynomial-time algorithm for minimizing submodular functions. *Journal of the ACM* **48** (2001) 761–777
13. Jeavons, P.: On the algebraic structure of combinatorial problems. *Theoretical Computer Science* **200** (1998) 185–204
14. Jeavons, P., Cohen, D., Cooper, M.: Constraints, consistency and closure. *Artificial Intelligence* **101**(1–2) (1998) 251–265
15. Jeavons, P., Cohen, D., Gyssens, M.: How to determine the expressive power of constraints. *Constraints* **4** (1999) 113–131
16. Narayanan, H.: *Submodular Functions and Electrical Networks*. North-Holland, Amsterdam (1997)
17. Queyranne, M.: Minimising symmetric submodular functions. *Mathematical Programming* **82** (1998) 3–12
18. Rossi, F., van Beek, P., Walsh, T., eds.: *The Handbook of Constraint Programming*. Elsevier (2006)