Alcom-FT Technical Report Series
ALCOMFT-TR-03-55

Multimorphisms: Classifying the Complexity of Soft Constraints*

David Cohen[1], Martin Cooper[2], Peter Jeavons[3], and Andrei Krokhin[4]

[1] Department of Computer Science, Royal Holloway, University of London, UK
[2] IRIT, University of Toulouse III, France
[3] Computing Laboratory, University of Oxford, UK
[4] Computer Science Department, University of Warwick, UK

Abstract. Over the past few years there has been considerable progress in methods to systematically analyse the complexity of classical (crisp) constraint satisfaction problems with specified constraint types. One very powerful theoretical development in this area links the complexity of a set of classical constraints to a corresponding set of algebraic operations, known as polymorphisms.
In this paper we begin a systematic investigation of the complexity of combinatorial optimisation problems expressed using various forms of soft constraints. We extend the notion of a polymorphism by introducing a more general algebraic operation, which we call a multimorphism. We show that a number of maximal tractable sets of soft constraints, both established and novel, can be characterised by the presence of particular multimorphisms.

Keywords: soft constraints, valued constraint satisfaction, combinatorial optimisation, submodular functions, tractability, polymorphism, multimorphism.

1 Introduction

In the standard constraint satisfaction framework a *constraint* is usually taken to be a predicate, or relation, specifying the allowed combinations of values for some fixed collection of variables: we will refer to such constraints here as *crisp* constraints. Problems with crisp constraints deal only with *feasibility*: no satisfying solution is considered better than any other.

A number of authors have suggested that the usefulness of the constraint satisfaction framework could be greatly enhanced by extending the definition of a constraint to include also *soft* constraints, which allow different measures of desirability to be associated with different combinations of values [1]. In this extended framework a constraint can be seen as a *function*, mapping each possible combination of values to a measure of desirability or undesirability. Problems with soft constraints deal with *optimisation* as well as feasibility: we seek an

* Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

assignment of values to all of the variables having the best possible overall combined measure of desirability.

Example 1. Consider an optimisation problem where we have $2n$ variables, $v_1, v_2, \ldots, v_{2n}$, and we wish to assign each variable an integer value in the range $1, 2, \ldots, n$, subject to the following restrictions:

- The value assigned to v_{2n} must be at least twice the value assigned to v_n.
- Each variable v_i should be assigned a value that is as close as possible to $i/2$.
- Each pair of variables v_i, v_{2i} should be assigned a pair of values that are as similar as possible.

To model this situation we might impose constraints as follows:

- A binary constraint on the pair v_n, v_{2n} specified by a function ζ, where $\zeta(x, y) = 0$ if $y \geq 2x$ and ∞ otherwise.
- A unary constraint on each v_i specified by a function ψ_i, where $\psi_i(x) = |x - i/2|^r$ for some $r \geq 1$.
- A binary constraint on each pair v_i, v_{2i} specified by a function δ_r, where $\delta_r(x, y) = |x - y|^r$ for some $r \geq 1$.

We would then seek an assignment to all of the variables which minimised the sum of these constraint functions,

$$\zeta(v_n, v_{2n}) + \sum_{i=1}^{2n} \psi_i(v_i) + \sum_{i=1}^{n} \delta_r(v_i, v_{2i}).$$

The cost of allowing additional flexibility in the specification of constraints, in order to model optimisation criteria as well as feasibility, is generally an increase in computational difficulty. For example, we establish below that the class of problems containing only unary constraints and a soft version of the equality constraint is NP-hard (see Example 5).

On the other hand, for certain types of soft constraint it is possible to solve the associated optimisation problems efficiently. For example, we establish below that optimisation problems of the form described in Example 1 can be solved in polynomial time (see Example 12).

In the case of crisp constraints there has been considerable progress in analyzing the complexity of different types of constraints. This work has led to the identification of a number of classes of constraints which are *tractable*, in the sense that there exists a polynomial time algorithm to determine whether or not any collection of constraints from such a class can be simultaneously satisfied [2, 10, 19, 27]. One powerful result in this area establishes that any tractable class of constraints over a finite domain must be preserved by a non-trivial algebraic operation, known as a *polymorphism* [3, 18, 19].

In the case of soft constraints there has not yet been any detailed investigation of the tractable cases, except for the special case of a two-valued domain [6].

In this paper we take the first step towards a systematic analysis of the complexity of soft constraints over arbitrary finite domains. To do this we generalise the algebraic ideas used to study crisp constraints, and introduce a new algebraic operation known as a *multimorphism*. Every soft constraint has an associated set of multimorphisms, and every multimorphism has an associated set of soft constraints. We show that, for several different types of multimorphism, the associated set of soft constraints forms a maximal tractable set. In other words, we show that several maximal tractable classes of soft constraints can be precisely characterised as the set of all soft constraints associated with a particular multimorphism.

The examples given below demonstrate that the framework we introduce here can be used to unify isolated results about tractable problem classes from many different application areas, as well as prompting the discovery of new tractable classes. For example, the notion of a multimorphism can be used to characterise tractable subproblems in all of the following areas: in the case of the SATISFIABILITY problem these include the HORN-SAT and 2-SAT subproblems [13]; in the case of the standard constraint satisfaction problem these include generalisations of HORN-SAT (such as the so-called 'max-closed' constraints [22, 19]), generalisations of 2-SAT (such as the so-called '0/1/all' or 'implicative' constraints [5, 17, 24]) and systems of linear equations [19]; in the case of the optimisation problem MAX-SAT these include the '0-valid' and '2-monotone' constraints [6]; in the case of optimisation problems over sets these include the submodular set functions [16, 26] and bisubmodular set functions [12].

2 Definitions

Several alternative mathematical frameworks for soft constraints have been proposed in the literature, including the very general frameworks of 'semi-ring based constraints' and 'valued constraints' [1]. For simplicity, we shall adopt the valued constraint framework here (although our results can easily be adapted to the semi-ring framework, for appropriate semi-ring structures).

In the valued constraint framework, a constraint is specified by a function which assigns a *cost* to each possible assignment of values. In general, costs may be chosen from any *valuation structure*, satisfying the following definition.

Definition 1. *A* ***valuation structure*** *is an ordered set V, with a minimum and a maximum element (denoted 0 and ∞), together with a commutative, associative binary* ***aggregation operator*** *(denoted $+$), such that for all $\alpha, \beta, \gamma \in V$*

$$\alpha + 0 = \alpha \tag{1}$$

$$\alpha + \gamma \geq \beta + \gamma \quad \textit{whenever} \quad \alpha \geq \beta. \tag{2}$$

For all of the examples given in this paper we shall use the valuation structure $\overline{\mathbb{R}^+}$, consisting of the non-negative real numbers together with infinity, with the usual ordering and the usual addition operation.

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

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

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

$$Cost_{\mathcal{P}}(t) = \sum_{\langle\langle v_1, v_2, \ldots, v_m \rangle, \phi \rangle \in C} \phi(t(v_1), t(v_2), \ldots, t(v_m)).$$

A ***solution*** *to* $\mathcal{P}$ *is an assignment with minimal cost, and the question is to find a solution.*

Example 2. For any standard constraint satisfaction problem instance $\mathcal{P}$ with crisp constraints, we can define a corresponding valued constraint satisfaction problem instance $\widehat{\mathcal{P}}$ in which the range of the cost functions of all the constraints is the set $\{0, \infty\}$. For each crisp constraint c of $\mathcal{P}$, we define a corresponding soft constraint $\widehat{c}$ of $\widehat{\mathcal{P}}$ with the same scope; the cost function of $\widehat{c}$ maps each tuple allowed by c to 0, and each tuple disallowed by c to ∞.

In this case the cost of an assignment t for $\widehat{\mathcal{P}}$ equals the minimal possible cost, 0, if and only if t satisfies all of the crisp constraints in $\mathcal{P}$.

Example 3. For any standard constraint satisfaction problem instance $\mathcal{P}$ with crisp constraints, we can define a corresponding valued constraint satisfaction problem instance $\mathcal{P}^{\#}$ in which the range of the cost functions of all the constraints is the set $\{0, 1\}$. For each crisp constraint c of $\mathcal{P}$, we define a corresponding soft constraint $c^{\#}$ of $\mathcal{P}^{\#}$ with the same scope; the cost function of $c^{\#}$ maps each tuple allowed by c to 0, and each tuple disallowed by c to 1.

In this case the cost of an assignment t for $\mathcal{P}^{\#}$ equals the number of crisp constraints in $\mathcal{P}$ which are violated by t. Hence a solution to $\mathcal{P}^{\#}$ corresponds to an assignment which violates the minimal number of constraints of $\mathcal{P}$.

The problem of finding a solution to a valued constraint satisfaction problem is an NP optimisation problem, that is, it lies in the complexity class NPO (see [6] for a formal definition of this class). It follows from Examples 2 and 3 that the general VCSP is NP-hard. To achieve more tractable versions of VCSP, we will now consider the effect of restricting the forms of cost function allowed in the constraints.

Definition 4. *Let D be a set and X a valuation structure. A* ***valued constraint language*** *over D with costs in X is defined to be a set of functions, Γ, such that each $\phi \in \Gamma$ is a function from D^m to X, for some $m \in \mathbb{N}$, where m is called the arity of ϕ.*

The class VCSP(Γ) *is defined to be to be class of all* VCSP *instances where the cost functions of all constraints lie in Γ.*

For any valued constraint language Γ, if every instance in VCSP(Γ) can be solved in polynomial time then we will say that Γ is **tractable**. On the other hand, if there is a polynomial-time reduction from some NP-complete problem to VCSP(Γ), then we shall say that VCSP(Γ) is **NP-hard**.

Example 4. Let Γ be any valued constraint language over D, where $|D| = 2$. In this case VCSP(Γ) is a *Boolean* soft constraint satisfaction problem.

If we restrict Γ even further, by only allowing functions with range $\{0, \infty\}$, as in Example 2, then each VCSP(Γ) corresponds precisely to a standard Boolean crisp constraint satisfaction problem. Such problems are sometimes known as Generalized Satisfiability problems [29, 13]. The complexity of VCSP(Γ) for such restricted sets Γ has been completely characterised, and the six tractable cases have been identified [29, 6].

Alternatively, if we restrict Γ by only allowing functions with range $\{0, 1\}$, as in Example 3, then each VCSP(Γ) corresponds precisely to a standard Boolean maximum satisfiability problem, in which the aim is to satisfy the maximum number of crisp constraints. Such problems are sometimes known as Max-Sat problems [6]. The complexity of VCSP(Γ) for such restricted sets Γ has been completely characterised, and the three tractable cases have been identified (see Theorem 7.6 of [6]).

We note, in particular, that when Γ contains just the single binary function ϕ_{XOR} defined by

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

then VCSP(Γ) corresponds to the Max-Sat problem for the exclusive-or predicate, which is known to be NP-hard (see Lemma 7.4 of [6]).

Example 5. Let Γ be a soft constraint language over D, where $|D| \geq 3$, and assume that Γ contains just the set of all unary functions, together with the single binary function ϕ_{EQ} defined by

$$\phi_{EQ}(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise.} \end{cases}$$

Even in this very simple case it can be shown that VCSP(Γ) is NP-hard, by reduction from the Minimum Multi-Terminal Cut problem [9]. (For details see [4].)

3 Reductions and Multimorphisms

Let Γ be a valued constraint language, and consider an arbitrary instance $\mathcal{P}$ in VCSP(Γ). If we choose a subset of the variables of $\mathcal{P}$ which is equal to the set of variables in the scope of some constraint of $\mathcal{P}$, then the values taken by those variables are explicitly constrained. What is more, if we choose *any* subset of the variables of $\mathcal{P}$, then the values may still be constrained *implicitly*, due to the combined effect of the constraints of $\mathcal{P}$. The cost function which describes this implicit constraint may or may not be an element of Γ, but can, in a sense, be expressed using elements of Γ.

The next two definitions formalise this idea of a function being *expressible* over a valued constraint language.

Definition 5. *For any* VCSP *instance* $\mathcal{P} = \langle V, D, C, X\rangle$*, and any tuple of variables* $W = \langle v_1, \ldots, v_k\rangle$*, the* ***cost function for*** $\mathcal{P}$ ***on*** W*, denoted* $\Phi_{\mathcal{P}}^{W}$*, is defined as follows:*

$$\Phi_{\mathcal{P}}^{W}(d_1, \ldots, d_k) = \min\{\,Cost_{\mathcal{P}}(t) \mid t : V \rightarrow D, \langle t(v_1), \ldots, t(v_k)\rangle = \langle d_1, \ldots, d_k\rangle\}$$

Definition 6. *A function* ϕ *is* ***expressible*** *over a valued constraint language* Γ *if there exists an instance* $\mathcal{P} = \langle V, D, C, X\rangle$ *in* VCSP(Γ) *and a list* W *of variables from* V *such that* $\phi = \Phi_{\mathcal{P}}^{W}$.

The notion of expressibility is a key tool in analysing the complexity of valued constraint languages, as the next result shows.

Proposition 1. *Let* Γ *be a valued constraint language, and* Γ' *a finite valued constraint language.*

If every $\phi \in \Gamma'$ *is expressible over* Γ*, then* VCSP(Γ') *is polynomial-time reducible to* VCSP(Γ).

Proof. Let $\mathcal{P} = \langle V, D, C, X\rangle$ be any instance in VCSP(Γ'), and let $c = \langle \sigma, \phi\rangle$ be a constraint in C. Since ϕ is expressible over Γ, there exists an instance $\mathcal{P}_{\phi}$ in VCSP(Γ), and a list of variables W of $\mathcal{P}_{\phi}$, such that $\Phi_{\mathcal{P}_{\phi}}^{W} = \phi$. Hence we can replace the constraint c with a copy of $\mathcal{P}_{\phi}$, where the variables in the scope σ are identified with the list of variables W, and the remaining variables of $\mathcal{P}_{\phi}$ are disjoint from V, to obtain a new problem instance $\mathcal{P}'$. Note that the solutions to $\mathcal{P}'$, when restricted to V, correspond precisely to the original solutions to $\mathcal{P}$.

By repeating this construction for each constraint c of $\mathcal{P}$, we can obtain an instance $\mathcal{P}''$ of VCSP(Γ). Since Γ' is finite, there is a bound on the size of the instances $\mathcal{P}_{\phi}$ used in the construction, and so the size of $\mathcal{P}''$ is bounded by a constant multiple of the size of $\mathcal{P}$. Hence we have described a polynomial-time reduction from VCSP(Γ') to VCSP(Γ).

It follows from Proposition 1 that finite valued constraint languages that can express precisely the same set of functions have the same complexity, up to polynomial-time reduction. Hence to analyse the complexity of a valued constraint language it may be sufficient to determine what functions can be expressed over that language. For example, the next result shows how this idea can be used to establish NP-hardness of a valued constraint language.

Corollary 1. *Let Γ be a valued constraint language over D, with costs in $\overline{\mathbb{R}^+}$.*

If there exist $d, d' \in D$, and $\alpha, \beta \in \overline{\mathbb{R}^+}$, with $\alpha < \beta < \infty$, such that the binary function ϕ_{XOR+} given by

$$\phi_{XOR+}(x,y) = \begin{cases} \alpha & \text{if } x \neq y \ \wedge \ x,y \in \{d,d'\} \\ \beta & \text{if } x = y \ \wedge \ x,y \in \{d,d'\} \\ \infty & \text{otherwise} \end{cases}$$

is expressible over Γ, then VCSP(Γ) *is NP-hard.*

Proof. Any instance of VCSP($\{\phi_{XOR+}\}$) must have a solution involving only the two values d and d', since all assignments involving any other values must have costs at least as high. Lemma 7.4 of [6] states that the two-valued problem VCSP($\{\phi_{XOR}\}$) is NP-hard, where ϕ_{XOR} is the Boolean exclusive-or function, as defined in Example 4. Since adding a constant to all cost functions, and scaling all costs by a constant factor, does not affect the difficulty of solving a VCSP instance, we conclude that VCSP($\{\phi_{XOR+}\}$) is also NP-hard.

For crisp constraints, it has been show that the expressive power of a set of relations is determined by certain algebraic invariance properties of those relations, known as *polymorphisms* [3, 19, 21, 20, 28]. The concept of a polymorphism is specific to *relations*, and cannot be applied directly to the *functions* in a valued constraint language. However, we now introduce a more general notion, which we call a *multimorphism*, which does apply directly to functions.

Definition 7. *Let D be a set, X a valuation structure, and $\phi : D^m \to X$ a function.*

We extend the definition of ϕ in the following way: for any positive integer k, and any list of k-tuples, $t_1, t_2, \ldots, t_m$, over D, we define

$$\phi(t_1, t_2, \ldots, t_m) = \sum_{i=1}^{k} \phi(t_1[i], t_2[i], \ldots, t_m[i])$$

We say that $F : D^k \to D^k$ is a ***multimorphism*** *of ϕ if, for any list of k-tuples $t_1, t_2 \ldots, t_m$ over D we have*

$$\phi(F(t_1), F(t_2), \ldots, F(t_m)) \leq \phi(t_1, t_2, \ldots, t_m).$$

Example 6. Let $D = \{0, 1, 2, \ldots, |D| - 1\}$ be a subset of the integers, and let $\phi : D^3 \to \overline{\mathbb{R}^+}$ be the linear function defined by $\phi(x, y, z) = ax + by + cz$, where a, b, c are positive constants.

Consider the function $F : D^2 \to D^2$ defined by $F(x, y) = \langle \min(x, y), \max(x, y) \rangle$.

For any list of pairs, t_1, t_2, t_3, over D we have

$$\begin{aligned}
&\phi(F(t_1), F(t_2), F(t_3)) \\
&= \phi(\langle \min(t_1[1], t_1[2]), \max(t_1[1], t_1[2])\rangle, \ldots, \langle \min(t_3[1], t_3[2]), \max(t_3[1], t_3[2])\rangle) \\
&= \phi(\min(t_1[1], t_1[2]), \min(t_2[1], t_2[2]), \min(t_3[1], t_3[2])) \\
&\quad + \phi(\max(t_1[1], t_1[2]), \max(t_2[1], t_2[2]), \max(t_3[1], t_3[2])) \\
&= (a \min(t_1[1], t_1[2]) + b \min(t_2[1], t_2[2]) + c \min(t_3[1], t_3[2])) \\
&\quad + (a \max(t_1[1], t_1[2]) + b \max(t_2[1], t_2[2]) + c \max(t_3[1], t_3[2]) \\
&= a(t_1[1] + t_1[2]) + b(t_2[1] + t_2[2]) + c(t_3[1] + t_3[2]) \\
&= \phi(t_1, t_2, t_3)
\end{aligned}$$

Hence F is a multimorphism of ϕ.

Example 7. Let R be a relation of arity m, and let ϕ_R be the function defined by

$$\phi_R(x_1, x_2, \ldots, x_m) = \begin{cases} 0 & \text{if } \langle x_1, x_2, \ldots, x_m\rangle \in R \\ \infty & \text{otherwise} \end{cases}$$

There is a close relationship between the *polymorphisms* of the relation R, as defined in [3, 20, 28], and the *multimorphisms* of the function ϕ_R.

For any polymorphism $f : D^k \to D$ of R, it is easy to show that the function $F : D^k \to D^k$ defined by

$$F(x_1, x_2, \ldots, x_k) = \langle f(x_1, x_2, \ldots, x_k), f(x_1, x_2, \ldots, x_k), \ldots, f(x_1, x_2, \ldots, x_k)\rangle$$

is a multimorphism of ϕ_R.

Furthermore, if $F : D^k \to D^k$ is a multimorphism of ϕ_R, then it is straightforward to check from the definitions that each of the k *component* functions, F_i, given by $F_i(x_1, x_2, \ldots, x_k) = F(x_1, x_2, \ldots, x_k)[i]$, is a polymorphism of R.

The following result means that multimorphisms have the key property that they extend to all functions expressible over a given language.

Theorem 1. *Let Γ be a valued constraint language, and F be a multimorphism of every function in Γ.*

If ϕ is expressible over Γ, then F is also a multimorphism of ϕ.

Proof. The proof of this result is a straightforward application of Definition 7 and Definition 6.

In the remainder of the paper we will show that a wide range of tractable optimisation problems are characterised by the presence of certain forms of multimorphism.

Example 8. For any set X, a function ψ defined on subsets of X is called a **submodular set function** [26] if for all subsets S_1 and S_2:

$$\psi(S_1 \cup S_2) + \psi(S_1 \cap S_2) \leq \psi(S_1) + \psi(S_2)$$

The problem of submodular set function minimisation consists in finding a subset S of X for which $\psi(S)$ is minimal. Such problems arise in a number of different contexts. For example, Cunningham [7] showed that finding the maximum flow in a network can be viewed as a special case of the general problem of submodular function minimisation.

By fixing an arbitrary order for the elements of X, we can associate each subset S of X with a tuple t_S of length $|X|$ over the set $\{0,1\}$, where $t_S[i] = 1$ if and only if S contains the ith element of X. Using this association, it is easy to show that a function ψ is a submodular set function if and only if the function ϕ given by $\phi(t_S) = \psi(S)$ has the multimorphism $F : \{0,1\}^2 \to \{0,1\}^2$ given by $F(x,y) = \langle \min(x,y), \max(x,y) \rangle$.

It has been known for a long time that real-valued submodular set functions can be minimised in polynomial time using the ellipsoid method [14]. Recently, several different strongly polynomial, combinatorial algorithms have been proposed for this problem [30, 15, 16, 11].

However, the best known polynomial-time bounds for general real-valued submodular set function minimisation are still rather high: the number of oracle calls has been shown to be $O(n^7)$, and the number of fundamental operations has been shown to be $O(n^8)$ [11].

Example 9. For any set X, a function ψ defined on pairs of disjoint subsets of X is called a **bisubmodular function** [12] if for all pairs (X_1, X_2) and (Y_1, Y_2) of disjoint subsets of X

$$\psi(\langle X_1, X_2 \rangle \sqcup \langle Y_1, Y_2 \rangle) + \psi(\langle X_1, X_2 \rangle \sqcap \langle Y_1, Y_2 \rangle) \leq \psi(\langle X_1, X_2 \rangle) + \psi(\langle Y_1, Y_2 \rangle)$$

where

$$\langle X_1, X_2 \rangle \sqcup \langle Y_1, Y_2 \rangle = \langle (X_1 \cup Y_1) \setminus (X_2 \cup Y_2), (X_2 \cup Y_2) \setminus (X_1 \cup Y_1) \rangle$$
$$\langle X_1, X_2 \rangle \sqcap \langle Y_1, Y_2 \rangle = \langle X_1 \cap Y_1, X_2 \cap Y_2 \rangle$$

It is known that an integer-valued bisubmodular function ψ can be minimised in $\mathrm{O}(n^5 \log M)$ time where M designates the maximum value of the function ψ [12].

By fixing an arbitrary order for the elements of X, we can associate each pair of disjoint subsets $\langle X_1, X_2 \rangle$ of X with a tuple $t_{\langle X_1, X_2 \rangle}$ of length $|X|$ over the set $\{0,1,2\}$, where $t_{\langle X_1, X_2 \rangle}[i] = 1$ if X_1 contains the ith element of X, $t_{\langle X_1, X_2 \rangle}[i] = 2$ if X_2 contains the ith element of X, and $t_{\langle X_1, X_2 \rangle}[i] = 0$ otherwise. Using this association, it is easy to check that a function ψ is a bisubmodular function if and only if the function ϕ given by $\phi(t_{\langle X_1, X_2 \rangle}) = \psi(\langle X_1, X_2 \rangle)$ has a multimorphism $F : \{0,1,2\}^2 \to \{0,1,2\}^2$ given by $F(x,y) = \langle \min_0(x,y), \max_0(x,y) \rangle$, where

$$min_0(x,y) = \begin{cases} min(x,y) & \text{if } \{x,y\} \neq \{1,2\} \\ 0 & \text{otherwise} \end{cases}$$

$$max_0(x,y) = \begin{cases} max(x,y) & \text{if } \{x,y\} \neq \{1,2\} \\ 0 & \text{otherwise} \end{cases}$$

Hence the bisubmodular functions defined in [12] are also characterised by the presence of a multimorphism.

4 Tractable Valued Constraint Languages

In this section we will present several tractable valued constraint languages. Some of these are translations of known tractable optimisation problems into the VCSP framework, and others are novel tractable classes. In all cases we are able to give a characterisation of the tractable language in terms of a single multimorphism. Hence, in all cases we have shown that the presence of a particular multimorphism is sufficient to guarantee tractability.

Definition 8. *Given a function $F : D^k \to D^k$, we will write Γ_F to denote the valued constraint language over D with costs in $\overline{\mathbb{R}^+}$, consisting of all functions for which F is a multimorphism.*

4.1 Constant multimorphisms

The first example we give is a rather straightforward family of tractable languages, characterised by the presence of a single, constant, multimorphism. Although the proof of tractability for this case is trivial, the proof that every language characterised by a constant multimorphism is a *maximal* tractable language is more interesting, and provides a simple example of the techniques we shall use for other cases.

Theorem 2. *Let D be a set, and let $F : D \to D$ be a constant function.*

1. *The set of functions Γ_F is a tractable valued constraint language.*
2. *Any valued constraint language Γ such that $\Gamma \supset \Gamma_F$ is NP-hard.*

Proof. Let d_F be the (constant) value of F.

1. Let ϕ be any function in Γ_F, and let m be the arity of ϕ. Since F is a multimorphism of ϕ, we have, for all $d_1, d_2, \ldots, d_m \in D$,

$$\phi(d_F, d_F, \ldots, d_F) \leq \phi(d_1, d_2, \ldots, d_m)$$

   Hence any instance $\mathcal{P}$ in VCSP(Γ_F) has a solution which assigns the value d_F to every variable, so VCSP(Γ_F) is tractable.
2. Now assume that $\Gamma \supset \Gamma_F$, and hence Γ contains a function ϕ of arity m such that F is not a multimorphism of ϕ. Hence there exist $d_1, d_2, \ldots, d_m \in D$ such that $\phi(d_1, d_2, \ldots, d_m) < \phi(d_F, d_F, \ldots, d_F)$.
   If $\phi(d_F, \ldots, d_F) < \infty$, then set $\mu = (\phi(d_F, \ldots, d_F) - \phi(d_1, \ldots, d_m))/2$, otherwise set $\mu = 1$. Choose i_0 such that $d_{i_0} \neq d_F$. Now define the functions δ and ψ as follows:

$$\delta(x_1, \ldots, x_m) = \begin{cases} 0 & \text{if} \quad \langle x_1, \ldots, x_m \rangle \in \{\langle d_1, \ldots, d_m \rangle, \langle d_F, \ldots, d_F \rangle\} \\ \infty & \text{otherwise} \end{cases}$$

$$\psi(x_1, x_2, x_3) = \begin{cases} \mu & \text{if} \quad \langle x_1, x_2, x_3 \rangle \in \{\langle d_{i_0}, d_{i_0}, d_{i_0} \rangle, \langle d_{i_0}, d_F, d_F \rangle\} \\ 0 & \text{otherwise} \end{cases}$$

   Note that $\delta, \psi \in \Gamma_F$.

We construct the instance $\mathcal{P} \in \mathrm{VCSP}(\Gamma)$ with variables

$$\{X_1, \ldots, X_m, Y_1, \ldots, Y_m, Z_1, \ldots, Z_m\}$$

and constraints

$$\begin{array}{ll} \langle\langle X_1, \ldots, X_m\rangle, \phi\rangle, & \langle\langle X_1, \ldots, X_m\rangle, \delta\rangle, \\ \langle\langle Y_1, \ldots, Y_m\rangle, \delta\rangle, & \langle\langle Z_1, \ldots, Z_m\rangle, \delta\rangle, \\ \langle\langle X_{i_0}, Y_{i_0}, Z_{i_0}\rangle, \psi\rangle & \end{array}$$

If we set $W = \langle Y_{i_0}, Z_{i_0}\rangle$, then it is straightforward to check that

$$\Phi_{\mathcal{P}}^{W}(x, y) = \begin{cases} \phi(d_1, d_2, \ldots, d_m) & \text{if } \ x \neq y \ \wedge \ x, y \in \{d_F, d_{i_0}\} \\ \mu + \phi(d_1, d_2, \ldots, d_m) & \text{if } \ x = y \ \wedge \ x, y \in \{d_F, d_{i_0}\} \\ \infty & \text{otherwise} \end{cases}$$

Hence, by Corollary 1, $\mathrm{VCSP}(\Gamma)$ is NP-hard.

Example 10. Recall from Example 4 that the MAX-SAT optimisation problem has just three maximal tractable classes, which are identified in [6]. Two of these can be characterised by having a constant function as a multimorphism; these are referred to in [6] as '0-valid' relations, and '1-valid' relations.

4.2 Min-max multimorphisms

The next example we give is a valued constraint language which can be defined on any finite ordered set D. This language is characterised by the presence of a single binary multimorphism, which we will call a **min-max** multimorphism. Languages with this multimorphism generalise the class of submodular set functions used in economics and operations research [26] (Example 8).

Theorem 3. *Let D be a finite ordered set, and let $F : D^2 \to D^2$ be the function defined by $F(d, d') = \langle \min(d, d'), \max(d, d')\rangle$.*

1. *The set of finite-valued functions in Γ_F is a tractable valued constraint language.*
2. *Any valued constraint language Γ such that $\Gamma \supset \Gamma_F$ is NP-hard.*

Proof.

1. To establish the tractability of the set of finite-valued functions in Γ_F, we show that this problem can be reduced to the problem of minimising a real-valued submodular set function [26] over a special family of sets known as a ring family [30]. This problem can then be solved in polynomial time using an algorithm due to Schrijver [30]. Details are omitted due to space restrictions.
2. Proof omitted.

Example 11. Recall from Example 4 that the MAX-SAT optimisation problem has just three maximal tractable classes, which are identified in [6]. One of these can be characterised by having a min-max multimorphism; this class is referred to in [6] as the class of '2-monotone' relations.

Example 12. It is a simple consequence of the definitions that every *unary* function has a min-max multimorphism.

We have recently shown that a problem instance $\mathcal{P} = (V, D, C, \overline{\mathbb{R}^+})$ involving unary and *binary* functions with a min-max multimorphism, including functions taking infinite values, can be solved in $O(|V|^3|D|^3)$ time [4]. Note that this compares very favourably with the best known complexity bound for optimising submodular set functions of arbitrary arity, as discussed in Example 8.

Let D be the set $\{0, 1, 2, \ldots, |D| - 1\}$, considered as a set of integers. The following *binary* functions all have a min-max multimorphism, and hence any VCSP instance involving constraints with cost functions of these forms can be solved in cubic time.

$$\phi_1(x, y) = \begin{cases} 0 & \text{if} \quad ax \leq by + c \text{ (for positive constants } a, b, c) \\ \infty & \text{otherwise} \end{cases}$$
$$\phi_2(x, y) = ax + by + c \text{ (for positive constants } a, b, c)$$
$$\phi_3(x, y) = \sqrt{x^2 + y^2}$$
$$\phi_4(x, y) = |x - y|^r \text{ (for } r \geq 1)$$

Using these observations we conclude that the discrete optimisation problem described in Example 1 can be solved in cubic time.

4.3 Max-max multimorphisms

The next example we give is a valued constraint language which can be defined on any finite ordered set D. This language is characterised by the presence of a single binary multimorphism, which we will call a **max-max** multimorphism. Languages with this multimorphism generalise the class of max-closed crisp constraints introduced and shown to be tractable in [22].

Theorem 4. *Let D be a finite ordered set, and let $F : D^2 \to D^2$ be the function defined by $F(d, d') = \langle \max(d, d'), \max(d, d') \rangle$.*

1. *The set of functions Γ_F is a tractable valued constraint language.*
2. *Any valued constraint language Γ such that $\Gamma \supset \Gamma_F$ is NP-hard.*

Proof. Omitted due to space restrictions.

Example 13. The constraint language CHIP incorporates a number of constraint solving techniques for arithmetic and other constraints. In particular it provides a constraint solver for a restricted class of crisp constraints over natural numbers, referred to as *basic constraints* [31]. These basic constraints are of two kinds which are referred to as “domain constraints” and “arithmetic constraints”. The domain constraints described in [31] are unary constraints which restrict the value of a variable to some specified finite subset of the natural numbers. The arithmetic constraints described in [31] are unary or binary constraints which have one of the following forms:

$$\begin{aligned}
aX &\neq b \\
aX &= bY + c \\
aX &\leq bY + c \\
aX &\geq bY + c
\end{aligned}$$

where variables are represented by upper-case letters, and constants by lower case letters. All constants are non-negative, and a is non-zero.

If we represent these crisp constraints as soft constraints where the range of the cost functions is the set $\{0, \infty\}$, as described in Example 2, then it is easy to verify that they all have a max-max multimorphism, and hence form a tractable soft constraint language, by Theorem 4.

Moreover, this tractable language can be extended, as shown in [22], to include functions corresponding to crisp constraints of the following forms, which also have a max-max multimorphism.

$$\begin{aligned}
a_1X_1 + a_2X_2 + \ldots + a_rX_r &\geq bY + c \\
aX_1X_2 \ldots X_r &\geq bY + c \\
(a_1X_1 \geq b_1) \vee (a_2X_2 \geq b_2) \vee \ldots \vee (a_rX_r \geq b_r) &\vee (aY \leq b)
\end{aligned}$$

Example 14. Let D be an ordered domain and X a valuation structure. A function $\phi : D^m \to X$ is called **antitone** if the value of $\phi(d_1, d_2, \ldots, d_m)$ does not increase when we increase any of the d_i.

All antitone functions have a max-max multimorphism, and hence form a tractable class. More importantly, they may be combined with the crisp constraints described in Example 13 to form a larger tractable class.

For example, let $D = \{0, 1, 2, \ldots, M\}$ be a subset of the integers, and let ϕ be the binary function defined by

$$\phi(x, y) = \begin{cases} (M - x)(M - y) & \text{if } x < y \\ \infty & \text{if } x \geq y \end{cases}$$

This function can be used to express a preference for larger values for x, y provided $x < y$. It is straightforward to check that it has a max-max multimorphism.

4.4 Majority/minority multimorphisms

The final example we give is a tractable valued constraint language which can be defined on any finite set D. This language is characterised by the presence of a single ternary multimorphism, which we will call a **majority/minority** multimorphism. Languages with this multimorphism generalise the class of bijective crisp constraints.

Theorem 5. *Let D be a finite set, and let $F : D^3 \to D^3$ be the function defined by $F(x,y,z) = \langle \mathrm{Maj}_1(x,y,z), \mathrm{Maj}_2(x,y,z), \mathrm{Min}_3(x,y,z)\rangle$ where*

$$\mathrm{Maj}_1(x,y,z) = \begin{cases} y & \textit{if} \quad y = z \\ x & \textit{otherwise.} \end{cases}$$
$$\mathrm{Maj}_2(x,y,z) = \begin{cases} x & \textit{if} \quad x = z \\ y & \textit{otherwise.} \end{cases}$$
$$\mathrm{Min}_3(x,y,z) = \begin{cases} x & \textit{if} \quad y = z \neq x \\ y & \textit{if} \quad x = z \neq y \\ z & \textit{otherwise.} \end{cases}$$

1. *The set Γ_F is a tractable valued constraint language.*
2. *Any valued constraint language Γ such that $\Gamma \supset \Gamma_F$ is NP-hard.*

Proof. Omitted due to space restrictions.

5 Conclusions

In this paper we have begun a systematic investigation of the complexity of the optimisation problems resulting from different forms of soft constraint. Since soft constraints are specified by functions, we have introduced an algebraic property of a function, which we call a multimorphism, and shown that in a range of cases the presence of such a property is sufficient to ensure tractability. Indeed, we have shown that the presence of a multimorphism precisely characterises a number of tractable problem classes that appear on the surface to be very different.

Further study is needed to determine whether the notion of a multimorphism exactly captures the expressive power, and hence the complexity, of soft constraints over finite domains. If this is true, then multimorphisms are likely to play a central role in the analysis of complexity for soft constraints, just as the related notion of a polymorphism does in the analysis of complexity for crisp constraints.

Other future developments to this work could include the study of approximability properties for optimisation problems involving soft constraints over arbitrary finite domains. This would build on and extend the detailed and successful investigation which has already been completed for the Boolean case [6].

References

1. S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4:199–240, 1999.
2. M. Bjäreland and P. Jonsson. Exploiting bipartiteness to identify yet another tractable subclass of CSP. In J. Jaffar, editor, *Principles and Practice of Constraint Programming—CP'99*, volume 1713 of *Lecture Notes in Computer Science*, pages 118–128. Springer-Verlag, 1999.

3. A.A. Bulatov, A.A. Krokhin, and P.G. Jeavons. Constraint satisfaction problems and finite algebras. In *Proceedings 27th International Colloquium on Automata, Languages and Programming, ICALP'00*, volume 1853 of *Lecture Notes in Computer Science*, pages 272–282. Springer-Verlag, 2000.
4. D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. A tractable class of soft constraints. Technical Report CSD-TR-02-14, Computer Science Department, Royal Holloway, University of London, Egham, Surrey, UK, December 2002.
5. M.C. Cooper, D.A. Cohen, and P.G. Jeavons. Characterising tractable constraints. *Artificial Intelligence*, 65:347–361, 1994.
6. N. Creignou, S. Khanna, and M. Sudan. *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.
7. W.H. Cunningham. Minimum cuts, modular functions,and matroid polyhedra. *Networks*, 15(2):205–215, 1985.
8. William H. Cunningham and Lawrence Tang. Optimal 3-terminal cuts and linear programming. In R.E. Barkard & G.J. Woeginger G. Cornuéjols, editor, *IPCO*, volume 1610, pages 114–125. Springer-Verlag, 1999.
9. E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.
10. T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal of Computing*, 28:57–104, 1998.
11. Lisa Fleischer and Satoru Iwata. Improved algorithms for submodular function minimization and submodular flow. In *Proceedings of the 32th Annual ACM Symposium on Theory of Computing*, pages 107–116, 2000.
12. Satoru Fujishige and Satoru Iwata. Bisubmodular function minimization. *Lecture Notes in Computer Science*, 2081:160–169, 2001.
13. M. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA., 1979.
14. M. Grötschel, L. Lovasz, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–198, 1981.
15. S. Iwata. A fully combinatorial algorithm for submodular function minimization. *Journal of Combinatorial Theory, Series B*, 84(2):203–212, 2002.
16. Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM*, 48(4):761–777, 2001.
17. P.G. Jeavons, D.A. Cohen, and M.C. Cooper. Constraints, consistency and closure. *Artificial Intelligence*, 101(1–2):251–265, 1998.
18. P.G. Jeavons, D.A. Cohen, and M. Gyssens. A test for tractability. In *Proceedings 2nd International Conference on Constraint Programming—CP'96 (Boston, August 1996)*, volume 1118 of *Lecture Notes in Computer Science*, pages 267–281. Springer-Verlag, 1996.
19. P.G. Jeavons, D.A. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44:527–548, 1997.
20. P.G. Jeavons, D.A. Cohen, and M. Gyssens. How to determine the expressive power of constraints. *Constraints*, 4:113–131, 1999.
21. P.G. Jeavons, D.A. Cohen, and J.K. Pearson. Constraints and universal algebra. *Annals of Mathematics and Artificial Intelligence*, 24:51–67, 1998.

22. P.G. Jeavons and M.C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2):327–339, 1995.
23. David R. Karger, Philip Klein, Cliff Stein, Mikkel Thorup, and Neal E. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 668–678. ACM Press, 1999.
24. L. Kirousis. Fast parallel constraint satisfaction. *Artificial Intelligence*, 64:147–160, 1993.
25. R. Mohr and G. Masini. Good dld discrete relaxation. In Y. Kodratoff, editor, *Proceedings 8th European Conference on Artificial Intelligence —ECAI'88*, pages 651–656. Pitman, 1988.
26. G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.
27. J.K. Pearson and P.G. Jeavons. A survey of tractable constraint satisfaction problems. Technical Report CSD-TR-97-15, Royal Holloway, University of London, July 1997.
28. R. Pöschel and L.A. Kalužnin. *Funktionen- und Relationenalgebren*. DVW, Berlin, 1979.
29. T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings 10th ACM Symposium on Theory of Computing, STOC'78*, pages 216–226, 1978.
30. A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *JCTB: Journal of Combinatorial Theory, Series B*, 80:346–355, 2000.
31. P. van Hentenryck, Y. Deville, and C-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.