The Complexity and Expressive Power of Valued Constraints

Stanislav Živný

Keble College, Oxford

Submitted for the degree of Doctor of Philosophy
Oxford University Computing Laboratory
Trinity Term 2009

The Complexity and Expressive Power of Valued Constraints

Stanislav Živný
Keble College, Oxford
Doctor of Philosophy, Trinity Term 2009

Abstract

This thesis is a detailed examination of the expressive power of valued constraints and related complexity questions. The valued constraint satisfaction problem (VCSP) is a generalisation of the constraint satisfaction problem which allows a variety of combinatorial optimisation problems to be described. Although most results are stated in this framework, they can be interpreted equivalently in the framework of, for instance, pseudo-Boolean polynomials, Gibbs energy minimisation or Markov Random Fields.

We take a result of Cohen, Cooper and Jeavons that characterises the expressive power of valued constraints in terms of certain algebraic properties, and extend this result by showing a new connection between the expressive power of valued constraints and linear programming. We prove a decidability result for related algebraic properties.

We consider various classes of valued constraints and the associated cost functions with respect to the question of which of these classes can be expressed using only cost functions of bounded arities. We identify the first known example of an infinite chain of classes of constraints with strictly increasing expressive power. We also present a full classification of various classes of constraints with respect to this problem.

We then study submodular constraints and cost functions. Submodular functions play a key role in combinatorial optimisation and are often considered to be a discrete analogue of convex functions. It has previously been an open problem whether all Boolean submodular cost functions can be decomposed into a sum of binary submodular cost functions over a possibly larger set of variables. This problem has been considered within several different contexts in computer science, including computer vision, artificial intelligence, and pseudo-Boolean optimisation. Using a connection between the expressive power of valued constraints and certain algebraic properties of cost functions, we answer this question negatively.

These results have several corollaries. First, we characterise precisely which submodular polynomials of arity 4 can be expressed by binary submodular polynomials. Next, we identify a novel class of submodular functions of arbitrary arities that can be expressed by binary submodular functions, and therefore minimised efficiently using a so-called expressibility reduction to the (s,t)-Min-Cut problem. More importantly, our results imply limitations on this kind of reduction and establish for the first time that it cannot be used in general to minimise arbitrary submodular functions. Finally, we refute a conjecture of Promislow and Young on the structure of the extreme rays of the cone of Boolean submodular functions.

Nothing can come of nothing.

William Shakespeare (1564–1616)

Acknowledgements

First of all, I would like to thank my supervisor Pete Jeavons. Pete suggested the study of the expressive power of valued constraints which has grown into this thesis. His academic input has shaped the direction of my work. I am tremendously grateful for his support and advice away from academic matters. Pete has always been around to help me not only with research problems, getting funding and missing definite articles, but also with getting around in Oxford, or just chatting about life. Thank you for all the encouragement!

Although not officially my second supervisor, Dave Cohen from Royal Holoway in London has always been one. His enthusiasm, support and many great ideas were crucial for completing my doctorate. Pete and Dave are excellent examples of brilliant teamwork: the combination of their individual skills has helped produce great results. Thank you for letting me be part of this.

I thank my colleagues from the Oxford Constraints research group: Martin Green, Chris Jefferson, Justyna Petke, Karen Petrie and András Salamon. In particular, talks with András, from whom I learnt that success is not only about good results, but more about communicating these results, and with Chris, who was always willing to explain anything I needed to understand, were extremely stimulating and helped me during my time at Oxford.

Martin Cooper visited our research group every summer and always came up with great research ideas. Thank you Martin. I also thank my coauthor Bruno Zanuttini.

My thanks go to Georg Gottlob, who served as my departmental advisor, and Stephan Kreutzer for many useful discussions and encouragement. I also thank other members of the Computing Laboratory at Oxford, namely Vince Bárány, Michael Benedikt, Chris Broadbent, Andrea Calì, Stephen Cameron, Stephen Clark, Matthew Hague, Elnar Hajiyev, Paul Hunter, Tom Melham, Andrzej Murawski, Luke Ong, Sebastian Ordyniak, Joël Ouaknine, Andy Twigg and Ben Worrell.

I appreciate support from, and interesting talks with, the participants of the DIMACS-RUTCOR Workshop on Boolean and Pseudo-Boolean Functions held in Memory of Peter L. Hammer in January 2009, namely Endre Boros, Jehoshua Bruck, Yves Crama, Martin Golumbic and Michel Minoux.

I learnt a lot during my studies at Charles University in Prague. I wish to thank my former supervisor Václav Koubek, mentor Antonín Kučera and many great teachers including Roman Barták, Ondřej Čepek, Dan Král', Jan Kratochvíl, Jaroslav Nešetřil, Petr Štěpánek and Jiří Wiedermann.

From my time at VU University in Amsterdam, I would like to thank András Kerekes for his friendship, and Leen Torenvliet and Femke van Raamsdonk for academic support.

From my time at Turku University in Finland, I thank Vesa Halava for support.

The UK Engineering and Physical Sciences Research Council supported me financially with a Research Studentship, and I am grateful to the Computing Laboratory and Keble College, Oxford, for financial support. I am grateful to Hertford College, Oxford, for appointing me a Lecturer in Computing in the last year of my DPhil studies. I am also grateful to University College, Oxford, for appointing me a Junior Research Fellow after completing my DPhil, thus making me feel safe and with a job while writing up this thesis.

I owe a lot to many friends I have made during my time at Oxford, including friends from Keble, rowing and volleyball. I would like to name at least the closest ones: Biljana, Efthymios, Piotr and Teresa - thank you for distraction from science:-)

Finally, I would like to thank my family back in the Czech republic: my twin sister Radka, and my mum for their support.

Last but not least, many thanks go to my significant other, Biying, who has been with me almost all my time at Oxford.

Declarations

I hereby certify that I have written this thesis entirely by myself. Parts of the thesis have appeared in the following papers which have been subject to peer review.

[CJŽ08] D.A. Cohen, P.G. Jeavons, and S. Živný. The Expressive Power of Valued Constraints: Hierarchies and Collapses. *Theoretical Computer Science*, 409(1):137–153, 2008.
Earlier version in *Proceedings of the 13th International Conference on Principles and Practice of Contraint Programming (CP'07)*, volume 4741 of *Lecture Notes in Computer Science*, pages 798–805. Springer, 2007.

[ZŽ09] B. Zanuttini and S. Živný. A Note on Some Collapse Results of Valued Constraints. *Information Processing Letters*, 109(11):534–538, 2009.

[ŽJ09a] S. Živný and P.G. Jeavons, Classes of Submodular Constraints Expressible by Graph Cuts. To appear in *Constraints*, 2009.
Earlier version in *Proceedings of the 14th International Conference on Principles and Practice of Contraint Programming (CP'08)*, volume 5202 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2008.

[ŽCJ09] S. Živný, D.A. Cohen, and P.G. Jeavons. The Expressive Power of Binary Submodular Functions. *Discrete Applied Mathematics*, 157(15):3347–3358, 2009.
Earlier version in *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science (MFCS'09)*, volume 5734 of *Lecture Notes in Computer Science*, pages 744–757. Springer, 2009.

Although all these papers have been written by me, with the help of Pete Jeavons, some of the results were obtained in collaboration. I would like to express here my thanks to my coauthors: Dave Cohen, Pete Jeavons and Bruno Zanuttini. Although not (yet) a coauthor of any of my papers, some of the work would not have been possible without the help of Martin Cooper.

Unless stated otherwise, all the Theorems, Lemmas, etc. in this thesis are new results. The beginning of each chapter lists where the new results have been published.

Contents

List of Figures

CHAPTER 1

Introduction

Theoretical computer science is like sex.
Sure, it may give some practical results,
but that's not why we do it.
Richard Feynman (1918–1988)
(adapted)

This chapter is an informal introduction to, and an overview of, theoretical computer science, constraint programming and constraint satisfaction problems. We will present several examples of important computational problems which can be formulated in the constraint framework, which is the framework this thesis deals with. All important concepts will be defined more formally in Chapter 2. At the end of this chapter, we will describe the organisation of this thesis together with the main contributions of the thesis.

It seems quite hard to describe what theoretical computer science, or TCS for short, is about. While some research in TCS does have an immediate impact on the way computers operate, a lot of the ideas are about various different topics that at first sight may seem to have little to do with computers. Indeed, the uniting theme of TCS research is hard to describe, but it seems to have something to do with studying how information can be manipulated and measuring the costs associated with manipulating information [Rao]. To understand the things we want to understand, we create mathematical models that describe the part of the world that we want to study. We then prove properties of these models, discovering facts that are true beyond any doubt about the mathematical models that we have designed.

This thesis deals with certain combinatorial optimisation problems. 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.

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 additional constraints can be *expressed* by a given valued constraint 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.

Constraint programming Constraint programming is a powerful paradigm for solving combinatorial search problems that draws on a wide range of techniques from artificial intelligence, operational research, algorithms, graph theory and elsewhere [RvBW06]. The basic idea in constraint programming is that the user states the constraints and a general purpose constraint solver is used to solve them. Constraints are specified by relations, and an instance of the CONSTRAINT SATISFACTION problem, or CSP for short, states which relations should hold among the given decision variables. More formally, a constraint satisfaction problem consists of a set of variables, each with some domain of values, and a set of relations on subsets of these variables.

Example 1.1 (Timetabling [RvBW06])**.** For example, in timetabling exams at a university, the decision variables might be the times and locations of the different exams, and the constraints might be on the capacity of each examination room (for example, we cannot timetable more students to sit exams in a given room at any one time than the room's capacity), and on the exams timetabled at the same time (for example, we cannot timetable two exams at the same time if they share students in common).

Constraint solvers take a real-world problem like this, represented in terms of decision variables and constraints, and find an assignment to all the variables that satisfies the constraints. Most CSP instances are solved by interleaving a backtracking search with a series of constraint propagation phases. A CSP instance can be made locally consistent by repeatedly removing unsupported values from the domains of its variables. This may allow us to reduce the domain of a variable after an assignment has been made in the backtracking search phase.

Extensions of the basic CSP framework may involve, for example, finding optimal solutions according to one or more optimisation criterion (for example, minimising the number of days over which exams need to be timetabled), finding all solutions, replacing (some or all) constraints with preferences, and considering a distributed setting where constraints are distributed among several agents. We refer the reader to the Handbook of Constraint Programming [RvBW06] and other standard textbooks [Apt03, Dec03] for more references and a more detailed discussion of the examples from this chapter.

Example 1.2 (Sudoku)**.** Various Japanese puzzles are good examples of problems which can be easily modelled and solved via constraint programming. Arguably one of the most well-known puzzles of this type is SUDOKU. In an instance of SUDOKU,

4	6				1			
		2		9	6			
	3						6	8
							3	5
			6		5			
7	1							
8	4						7	
			5	1		9		
			3			2	4	

Figure 1.1: A SUDOKU instance (Example 1.2).

the aim is to fill in a 9×9 grid of squares with the digits $1, \ldots, 9$ in such a way that each digit occurs exactly once in each row, each column, and each of 9 specified 3×3 sub-grids. Each specific instance of SUDOKU has a selection of grid entries already filled-in, and the aim is to fill in the remaining entries (see Figure 1.1).

One way to model this problem as a CSP instance is to choose the set of variables to be the 81 grid squares. Each variable has as its domain $\{1, \ldots, 9\}$, except for the pre-selected grid squares whose domain consists of a single number. There are 27 ALL-DIFFERENT constraints: 9 on rows, 9 on columns, and 9 on the specified 3×3 sub-grids. As the name suggests, the ALL-DIFFERENT constraint is satisfied if all its arguments (in this case, 9) are given different values.

Constraint satisfaction As mentioned above, a CSP instance consists of a set of variables and a set of constraints imposed on these variables. Each constraint consists of a scope (a list of variables the constraint restricts), and a relation which specifies which combinations of values are allowed. The goal is to find an assignment of values to the variables such that all constraints are satisfied. One of the strengths of the CSP framework is that it provides a unifying framework for various classes of problems that have been studied independently before. By specifying what the domains and constraints are (for instance, domains can be finite, infinite, discrete,...), one can obtain different classes of problems. Here are some other examples (more examples can be found in Chapter 2).

Example 1.3 (Acyclicity)**.** Given a directed graph G, the question of whether G is acyclic can be modelled as a CSP instance as follows: variables correspond to the vertices of G, the domain of every variable is the set of natural numbers $\mathbb{N}$, and every arc (x, y) of G represents a constraint $x < y$, where $<$ is the usual "smaller than" ordering on natural numbers.

Example 1.4 (Linear Inequalities)**.** Any system of linear inequalities can be modelled as a CSP instance with the same variables; the domain of all variables is the set of real numbers $\mathbb{R}$, and constraints are of the form $a_1x_1 + \ldots + a_nx_n \leq b$, for some $a_1, \ldots, a_n, b \in \mathbb{Q}$.

Example 1.5 (Diophantine Equations)**.** Hilbert's 10th problem asks for an algorithm that decides whether a given system of polynomial equations with integer coefficients (a *diophantine* equation system) has an integer solution. This can be modelled as a CSP instance with variables $x_1, \ldots, x_n$, each with the domain $\mathbb{Z}$, and constraints of the form $ax + by + cz = d$, or $x * y = z$, for $a, b, c, d \in \mathbb{Z}$. Matiyasevič has shown that this problem is undecidable [Mat70].

Example 1.6 (Graph k-Colouring)**.** Given an undirected graph G and a natural number k, the k-Colouring problem asks whether the vertices of G can be assigned colours such that adjacent vertices are assigned different colours and at most k different colours are used in total. This problem can be easily modelled as a CSP instance as follows: variables correspond to the vertices of G, the domain of every variable is the set $\{1, \ldots, k\}$, and every edge (u, v) of G represents the $u \neq v$ constraint; that is, the binary disequality relation on a k-element set.

Example 1.7 (Satisfiability)**.** The standard propositional satisfiability problem for ternary clauses, 3-Sat, consists in determining whether it is possible to satisfy a Boolean formula given as a conjunction of ternary clauses. This can be viewed as a CSP instance, where clause $C = (l_1 \vee l_2 \vee l_3)$ corresponds to the constraint with the relation $\{0,1\}^3 \setminus \{\langle a_1, a_2, a_3\rangle\}$, where $a_i = 1$ if l_i is negated in C, and $a_i = 0$ otherwise, $1 \leq i \leq 3$. For instance, a clause $(x_1 \vee \neg x_2 \vee x_3)$ would correspond to the constraint $\langle\langle x_1, x_2, x_3\rangle, R\rangle$, where $\langle x_1, x_2, x_3\rangle$ is the scope of the constraint, and $R = \{0,1\}^3 \setminus \{\langle 0, 1, 0\rangle\}$ is the relation of the constraint.

To name just a few of the areas where the CSP has been applied, let us mention artificial intelligence (temporal and spatial reasoning) [BK08b], type systems for programming languages (set constraints), computational linguistics (tree description languages), computational biology (phylogenetic reconstruction) [RvBW06], database theory (conjunctive query containment) [SGG08], graph theory (H-colouring, graph partition problems) [HN04], computer algebra (polynomial equations, polynomial inequalities) [BNvO09], operational research (linear programming, integer programming) [ABKW08], Boolean satisfiability [CKS01], complexity theory [BKJ05] and many others [RvBW06].

From these examples it is clear that many NP-complete problems and many other problems can be formulated as CSP instances. Considerable effort has therefore been invested in identifying restricted classes of the CSP which are tractable. For Boolean problems such as those in Example 1.7, where the decision variables have just two possible values, Schaefer's dichotomy theorem gives an elegant characterisation of the six classes of relations that lead to tractable problem classes [Sch78]. Schaefer's result can be considered the first dichotomy result on the complexity of the CSP.

It appears to be considerably more difficult to characterise tractable classes for non-Booleans domains. Research has typically focused on two special forms of tractability: tractable languages (where the relations are fixed but they can be combined in any way) [CJ06], and tractable constraint (hyper)graphs (where the way constraints interact is restricted but any sort of relation can be used) [SGG08].

Expressibility In any CSP instance, the variables listed in the scope of each constraint are *explicitly* constrained. Moreover, if we choose *any* subset of all variables of the instance, then their values are constrained *implicitly* in the same way, due to the combined effect of the constraints. This motivates the concept of *expressibility*, which is illustrated in the next two examples.

Example 1.8. Consider a CSP instance consisting of four variables x_1 through x_4, each with domain $\{1, 2, 3\}$. The constraints correspond to solid edges in Figure 1.2. Here $\neq_3$ denotes the binary disequality relation over a 3-element set; that is, $\neq_3$ is

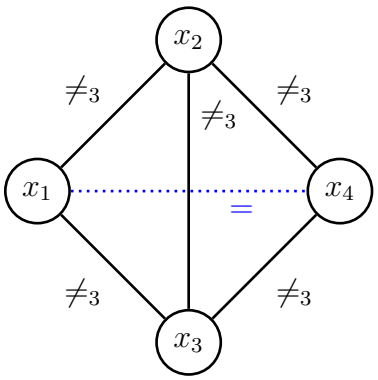

Figure 1.2: Disequalities express equality (Example 1.8).

the set $\{\langle i, j\rangle \mid 1 \leq i \neq j \leq 3\}$. It is easy to check that any assignment of values from $\{1, 2, 3\}$ to the variables x_1 through x_4, which satisfies all constraints, has to assign the same value to the variable x_1 and x_4. Hence the variables x_1 and x_4 are *implicitly* constrained by the equality relation. This shows that $=$ is expressible by $\neq_3$.

The notion of expressibility has been a key component in the complexity analysis of the CSP [JCG97, Jea98, BKJ05, LT09]. It was also a major tool in the complexity analysis of a wide variety of Boolean constraint problems carried out by Creignou et al. [CKS01], where it was referred to as *implementation*. Expressibility is a particular form of problem reduction: if a constraint can be expressed in a given constraint language, then it can be added to the language without changing the computational complexity of the associated class of problems.

Feder and Vardi observed that the CSP is equivalent to the homomorphism problem between relational structures [FV98] (this was independently discovered in [Jea98]). The notion of expressibility corresponds to expressibility using conjunction and existential quantification (*primitive positive formulas*). These connections

have lead to connecting the CSP with other fields of mathematics and computer science including, for instance, graph theory [HN04] and logic [KV07a]

Valued constraints The CSP framework deals with decision problems only, and therefore the constraints in a CSP instance are called *hard* constraints as they cannot be violated. A number of extensions have been added to the basic CSP framework to deal with questions of optimisation [RvBW06]. This thesis works with one of the very general extensions of the CSP framework, called the VALUED CONSTRAINT SATISFACTION problem, or VCSP for short.

Informally, a VCSP instance consists of a set of variables, a set of possible values, and a set of (soft) constraints. Each constraint has an associated cost function which assigns a cost (or a degree of violation) to every possible tuple of values for the variables in the scope of the constraint. The goal is to find an assignment of values to all of the variables which has the minimum total cost. Note that the CSP model is a special case of the VCSP where the range of all cost functions is $\{0, \infty\}$; in other words, all cost functions are just relations.

The notion of expressibility naturally extends from the CSP framework to the VCSP framework: conjunction is replaced by addition, and projection is replaced by minimisation.

Example 1.9. Let SUM be the ternary soft constraint which returns as a cost value the sum of its arguments. Consider the VCSP instance $\mathcal{P}_2$ with variables v_1, v_2, and v_3, each with the domain $\mathbb{N} = \{1, 2, \ldots\}$, and two constraints: $\text{SUM}(v_1, v_2, v_3)$, and $v_1 \neq v_3$ (see Figure 1.3). Here $\neq$ is the disequality relation over natural numbers, and can be also viewed as the cost function $\neq (x, y) = 0$ if $x \neq y$, and ∞ otherwise.

Now the variables v_1 and v_2 are (implicitly) constrained by the binary soft constraint with cost function ϕ such that $\phi(x_1, x_2) = x_1 + x_2 + 1$ if $x_1 \neq 1$, and $\phi(1, x_2) = x_2 + 3$. The intuition is that if $v_1 \neq 1$, then the assignment to v_1, v_2 can be completed with $v_3 = 1$, and otherwise it can be completed with $v_3 = 2$ (since 1 violates $v_1 \neq v_3$ and thus yields an infinite cost, which is bigger than the finite cost obtained by assigning $v_3 = 2$). Hence ϕ is expressible by the constraints SUM and $\neq$.

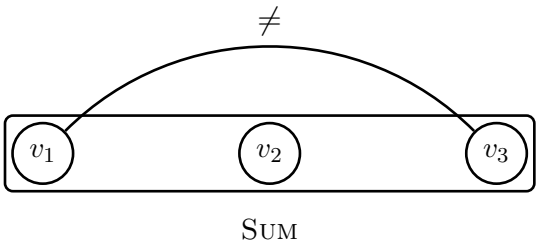

Figure 1.3: Instance $\mathcal{P}_2$ (Example 1.9).

Example 1.10. Consider the following relation $R_{\leq 1}$ defined as

$$R_{\leq 1} \;=\; \{\langle a, b\rangle \mid a, b \in \mathbb{Q}, a - b \leq 1\}.$$

Let $\mathcal{P}_1$ be the VCSP instance with variables v_1 through v_4, each with the domain $\mathbb{Q}$, and constraints illustrated in Figure 1.4. It is not difficult to see that the variables

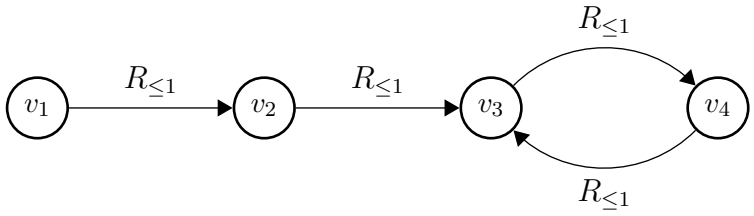

Figure 1.4: Instance $\mathcal{P}_1$ (Example 1.10).

v_1 and v_3 are constrained by the relation

$$R_{\leq 2} \;=\; \{\langle a, b\rangle \mid a, b \in \mathbb{Q}, a - b \leq 2\}.$$

Similarly, it is easy to see that the variables v_3 and v_4 are constrained by the relation

$$R_{-1\leq 1} \;=\; \{\langle a, b\rangle \mid a, b \in \mathbb{Q}, -1 \leq (a - b) \leq 1\}.$$

In general, $R_{\leq 1}$ can express any relation of the form

$$R_{-m\leq n} \;=\; \{\langle a, b\rangle \mid a, b \in \mathbb{Q}, -m \leq (a - b) \leq n\},$$

where m and n are positive integers or infinity.

This thesis

In view of Examples 1.8, 1.9 and 1.10, we study the expressive power of various classes of valued constraints, that is, the question of which other valued constraints can be expressed over these classes.

Contributions This thesis thoroughly explores the expressive power of valued constraints and makes significant contributions. It can been seen as a follow-up to work started in [CCJ06] and [CCJK06].

We investigate various algebraic properties of valued constraints in order to understand the expressive power of classes of relations and functions. We show that this so-called algebraic approach, traditionally used to separate tractable problems from intractable problems, is also useful for other questions, for instance, for finding boundaries to the applicability of certain algorithmic techniques.

Using this approach, we have been able to resolve some open problems, such as the question of expressibility of all Boolean submodular constraints by binary submodular constraints.

Thesis structure In Chapter 2, we formally define the Valued Constraint Satisfaction framework and other concepts used in this thesis. We give many examples of well-known problems which have been studied independently in various contexts of mathematics and computer science, and which can be naturally formulated in the VCSP framework.

Chapter 3 extends results from [CCJ06] and provides a new link between the problem of the expressive power of valued constraints and linear programming. We also prove several results on the algebraic properties of valued constraints and prove a decidability result dual to results in [CCJ06].

Chapter 4 deals with the expressive power of fixed-arity languages. We present a full classification of various classes of constraints with respect to this problem. We identify the first known example of an infinite chain of classes of constraints with strictly increasing expressive power.

Chapter 5 presents results on the expressive power of submodular constraints. We show new classes of submodular constraints which are expressible by binary submodular constraints, and also present new and simpler proofs of some known results.

Chapter 6 generalises some previously known results on the expressive power of Boolean submodular constraints. In particular, we show that not all submodular constraints are expressible by binary submodular constraints. Furthermore, we present some consequences of our results, and link our results to other previously studied problems.

In Chapter 7, we summarise the results obtained in this thesis, and discuss directions for future research.

CHAPTER 2

Background

The highest technique is to have no technique.
Bruce Lee (1940–1973)

In this chapter, we introduce the necessary background on valued constraints and submodular functions. In Section 2.1, we define the valued constraint satisfaction problem, present basic properties of this framework, and survey a list of well-known problems which can be easily described in this framework. In Section 2.3, we introduce the concept of expressibility for valued constraints. In Section 2.4, we present algebraic properties of valued constraints which are related to expressibility of valued constraints. Finally, in Section 2.5, we define submodular functions, and the submodular function minimisation problem.

2.1 Valued constraints

A major area of investigation in artificial intelligence is the Constraint Satisfaction problem (CSP) [Mon74]. The CSP is a general framework which can be used to model many different problems [CKS01, Dec03, RvBW06, Jea09]. The key idea underlying the CSP is to solve a problem by stating constraints representing requirements about the problem and, then, finding a solution satisfying all the constraints. However, the CSP model considers only the feasibility of satisfying a collection of simultaneous requirements, so-called *hard constraints*.

A number of extensions have been added to the basic CSP framework to deal with questions of optimisation, including semi-ring CSPs, valued CSPs, soft CSPs and weighted CSPs. These extended frameworks can be used to model a wide range of discrete optimisation problems [SFV95, BFM+99, RvBW06], including standard problems such as (s,t)-Min-Cut, Max-Sat, Max-Ones Sat, Max-CSP [CKS01, CCJK06], and Min-Cost Homomorphism [GRYT06].

The differences between the various general frameworks for soft constraints, such as valued constraints or semi-ring constraints, are not relevant for our purposes. The semi-ring CSP framework is slightly more general,[1] but the valued CSP framework is sufficiently powerful to model a wide range of optimisation problems. Hence we will simply focus on this one very general framework, the VALUED CONSTRAINT SATISFACTION problem (VCSP).

Informally, in the VCSP, an instance consists of a set of variables, a set of possible values, and a set of (soft) constraints. Each constraint has an associated cost function which assigns a cost (or a degree of violation) to every possible tuple of values for the variables in the scope of the constraint. The goal is to find an assignment of values to all of the variables which has the minimum total cost.

Remark 2.1.1. We remark that infinite costs can be used to indicate infeasible assignments (hard constraints), and hence the VCSP framework includes the standard CSP framework as a special case and is equivalent to the CONSTRAINT OPTIMISATION problem framework, COP, which is widely used in practice [RvBW06].

Remark 2.1.2. The VCSP framework is equivalent to *graphical models* [DM07].

Notation 2.1.3. We denote by $\mathbb{R}$ the set of all real numbers, and by $\overline{\mathbb{R}}$ the set of all real numbers together with (positive) infinity. Members of $\overline{\mathbb{R}}$ are called *costs*. We also denote by $\mathbb{R}_+$ the set of all nonnegative real numbers, and by $\overline{\mathbb{R}}_+$ the set of all nonnegative real numbers with (positive) infinity.

Remark 2.1.4. In order to avoid difficulties with representation issues for transcendental numbers such as π or e, we implicitly restrict the set of reals to the set of algebraic reals; that is, reals which are roots of non-zero polynomials in one variable with rational (or equivalently, integer) coefficients. In fact, as is common in computer science, artificial intelligence, and operational research, in practice the only numbers we make use of are rationals.

Notation 2.1.5. For any fixed set D, a function ϕ from D^m to $\overline{\mathbb{R}}$ will be called a *cost function* on D of arity m. D is called a *domain*, and in this thesis we will only deal with finite domains. If the range of ϕ lies entirely within $\mathbb{R}$, then ϕ is called a *finite-valued* cost function. If the range of ϕ is $\{0, \infty\}$, then ϕ is called a *crisp* cost function. If the range of a cost function ϕ includes both nonzero finite costs and infinity, we emphasise this fact by calling ϕ a *general* cost function.

Note that with any *relation* R on 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 a relation R_ϕ defined as $\langle x_1, \ldots, x_m \rangle \in R_\phi \Leftrightarrow \phi(x_1, \ldots, x_m) < \infty$, or equivalently an m-ary crisp cost function defined by:

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

[1] The main difference between semi-ring CSPs and valued CSPs is that costs in valued CSPs represent violation levels and have to be totally ordered, whereas costs in semi-ring CSPs represent preferences and might be ordered only partially.

Notation 2.1.6. We call $\mathsf{Feas}(\cdot)$ the *feasibility operator*.

In view of the close correspondence between crisp cost functions and relations we shall use these terms interchangeably in the rest of the thesis.

Definition 2.1.7 (VCSP)**.** An instance $\mathcal{P}$ of the VALUED CONSTRAINT SATISFACTION problem, VCSP, is a triple $\langle V, D, \mathcal{C}\rangle$, where V is a finite set of *variables*, which are to be assigned values from the set D, and $\mathcal{C}$ is a set of *valued constraints*. Each $c \in \mathcal{C}$ is a pair $c = \langle \mathbf{x}, \phi\rangle$, where $\mathbf{x}$ is a tuple of variables of length m, called the *scope* of c, and $\phi : D^m \to \overline{\mathbb{R}}$ is a cost function. An *assignment* for the instance $\mathcal{P}$ is a mapping s from V to D. We extend s to a mapping from V^k to D^k on tuples of variables by applying s componentwise. We denote by $\mathcal{A}$ the set of all assignments. The *cost* of an assignment s is defined as follows:

$$\mathsf{Cost}_{\mathcal{P}}(s) \stackrel{\text{def}}{=} \sum_{\langle \mathbf{x},\phi\rangle \in \mathcal{C}} \phi(s(\mathbf{x})).$$

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

Definition 2.1.8 (CSP)**.** An instance $\mathcal{P}$ of the CONSTRAINT SATISFACTION problem [RvBW06] is a VCSP instance where all cost functions are crisp, that is, relations. The task of finding an assignment with minimum cost amounts to testing whether all constraints can be satisfied (zero cost) or not (infinite cost).

Remark 2.1.9. In the original, more general, definition of the VCSP [BFM+99], costs were allowed to lie in any positive tomonoid S called a *valuation structure*.[2] Under the additional assumptions of discreteness and the existence of a partial inverse operation, it has been shown [Coo05] that such a structure S can be decomposed into independent positive tomonoids, each of which is isomorphic to a subset of $\overline{\mathbb{R}}_+$ with the operation being either standard addition, $+$, or bounded addition, $+_k$, where $a +_k b = \min(k, a+b)$. Therefore, using $\overline{\mathbb{R}}_+$ instead of an arbitrary valuation structure, we do not restrict ourselves too much. Moreover, using costs from $\overline{\mathbb{R}}_+$ and combining them using standard addition is standard in operational research.

In this thesis, we also allow (finite) negative costs, that is, costs from $\overline{\mathbb{R}}$ rather than just from $\overline{\mathbb{R}}_+$. As will be discussed in Section 2.3, we care about expressibility up to additive and multiplicative constants, and hence this does not give us anything new. However, this flexibility allows us to use standard examples and functions from the literature, and hence relate our work better to other results in the literature.

We show now that many classical problems can be formulated as subproblems of the VCSP.

Example 2.1.10 (CSP)**.** For any instance of the classical constraint satisfaction problem $\mathcal{P} = \langle V, D, \mathcal{C}\rangle$, we define a corresponding valued constraint satisfaction problem instance $\mathcal{P}' = \langle V, D, \mathcal{C}'\rangle$. For each constraint $\langle \sigma, R\rangle \in \mathcal{C}$ we define a cost function

[2] 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 $\alpha \geq \beta$.

ϕ_R and set $\mathcal{C}' = \{\langle \sigma, \phi_R \rangle | \langle \sigma, R \rangle \in \mathcal{C}\}$. The cost function σ_R maps each tuple allowed by R to 0, and each tuple disallowed by R to ∞. Any solution s to $\mathcal{P}'$ has cost 0 if, and only, if s satisfies all the constraints of $\mathcal{P}$.

Example 2.1.11 (Boolean Conjunctive Query Evaluation)**.** It is well known that certain fundamental problems in database theory such as Boolean Conjunctive Query Evaluation and Conjunctive Query Containment are equivalent to the CSP [FV98, KV00, FG06, GS08, SGG08], and hence can be formulated in the VCSP by Example 2.1.10.

Example 2.1.12 (Min-CSP/Max-CSP)**.** An instance $\mathcal{P}$ of the Maximum Constraint Satisfaction problem [CKS01] is an instance of the CSP with the goal to maximise the number of satisfied constraints. In the weighted version, each constraint has a non-negative weight and the goal is to maximise the weighted number of satisfied constraints. We shall denote by Max-CSP the more general weighted version.

Maximising the weighted number of satisfied constraints is the same as minimising the weighted number of unsatisfied constraints (Min-CSP).[3] Hence for any instance $\mathcal{P}$ of the Max-CSP or Min-CSP, we can define a corresponding VCSP instance $\mathcal{P}'$ in which a constraint c with weight w is associated with a cost function which maps tuples allowed by c to 0 and tuples disallowed by c to w.

Example 2.1.13 (Max-Ones)**.** Max-Ones is an extension of the Boolean CSP framework in which the goal is to satisfy all given constraints and simultaneously maximise the number of variables assigned the value 1 [CKS01]. In the weighted version each variable has a non-negative weight and the goal is to maximise the weighted number of variables assigned the value 1. We shall denote by Max-Ones the more general weighted version.

Similarly to the Max-CSP from Example 2.1.12, maximising the weighted number of variables assigned the value 1 is the same as minimising the weighted number of variables assigned the value 0. Hence for any instance $\mathcal{P}$ of Max-Ones, we can define an instance $\mathcal{P}'$ of the VCSP which has the same variables, domain and constraints as $\mathcal{P}$ with additional unary constraints: on a variable with weight w, we impose a unary constraint with the cost function μ defined as $\mu(0) = w$ and $\mu(1) = 0$.

Example 2.1.14 (Min-Ones)**.** Min-Ones is an extension of the Boolean CSP framework in which the goal is to satisfy all given constraints and simultaneously minimise the number of variables assigned the value 1 [CKS01]. In the weighted version each variable has a non-negative weight and the goal is to minimise the weighted number of variables assigned the value 1. We shall denote by Min-Ones the more general weighted version.

For any instance $\mathcal{P}$ of Min-Ones, we can define an instance $\mathcal{P}'$ of the VCSP which has the same variables, domain and constraints as $\mathcal{P}$ with additional unary constraints: on a variable with weight w, we impose a unary constraint with the cost function μ defined as $\mu(0) = 0$ and $\mu(1) = w$.

[3] This is true for optimal solutions. However, if we are interested in approximability results, this statement is not true even over Boolean domains, see [CKS01].

Example 2.1.15 ((s,t)-MIN-CUT)**.** Let $G = \langle V, E\rangle$ be a directed weighted graph such that for every $(u,v) \in E$ there is a weight $w(u,v) \in \overline{\mathbb{R}}_+$ and let $s,t \in V$ be the source and target nodes. An (s,t)-*cut* C is a subset of vertices V such that $s \in C$ but $t \notin C$. The weight, or the size, of an (s,t)-cut C is defined as $\sum_{(u,v)\in E, u\in C, v\notin C} w(u,v)$. The (s,t)-MIN-CUT problem consists in finding a minimum-weight (s,t)-cut in G.

We can formulate the search for a minimum-weight (s,t)-cut in G as a VCSP instance. For a fixed weight $w \in \overline{\mathbb{R}}_+$, we define

$$\lambda_w(x,y) \stackrel{\text{def}}{=} \begin{cases} w & \text{if } x = 0 \text{ and } y = 1, \\ 0 & \text{otherwise.} \end{cases}$$

For a fixed value $d \in \{0,1\}$ and a cost $c \in \overline{\mathbb{R}}_+$, we define

$$\mu_c^d(x) \stackrel{\text{def}}{=} \begin{cases} c & \text{if } x = d, \\ 0 & \text{if } x \neq d. \end{cases}$$

We denote by Γ_{cut} the set of cost functions λ_w and μ_c^d.

Let $\mathcal{P} = \langle V, \{0,1\}, \{\langle\langle u,v\rangle, \lambda_{w(u,v)}\rangle \mid (u,v) \in E\} \cup \{\langle s, \mu_\infty^1\rangle, \langle t, \mu_\infty^0\rangle\}\rangle$.

The unary constraints ensure that the source and target nodes take the values 0 and 1, respectively. Therefore, a minimum-weight (s,t)-cut in G corresponds to the set of variables assigned the value 0 in some solution to $\mathcal{P}$.

2.2 Complexity of VCSP

For each valued constraint satisfaction problem there is a corresponding decision problem in which the question is to decide whether there is a solution with cost lower than some given threshold value. It is clear from Example 2.1.10 that there is a polynomial-time reduction from the CSP to this decision problem. Since the CSP is known to be NP-complete [MF93], it follows that the VCSP is NP-hard.

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. Informally, NPO consists of function problems of the form "find an assignment of the variables $x_1, \ldots, x_k$ which minimises a cost function $\phi(x_1, \ldots, x_k)$, where ϕ is computable in polynomial time", see [ACG+99] for a formal definition of NPO. The VCSP framework is powerful enough to describe many NPO-complete problems, for instance MAX-ONES from Example 2.1.13, also known as MAXIMUM WEIGHTED SATISFIABILITY [ACG+99].

One significant line of research on the VCSP is to identify restrictions which ensure that instances are solvable in polynomial time. There are two main types of restrictions that have been studied in the literature.

First, we can limit the *structure* of the instances, in the following sense. With any instance $\mathcal{P}$ of the VCSP, we can associate a hypergraph $\mathcal{H}_\mathcal{P}$ whose vertices are the variables of $\mathcal{P}$, and whose hyperedges correspond to the scopes of the constraints of $\mathcal{P}$. The hypergraph $\mathcal{H}_\mathcal{P}$ is called the structure of $\mathcal{P}$, and is also know as the *constraint*

network [Dec03]. A number of results concerning restrictions to the structure of problem instances that are sufficient to ensure tractability have been obtained for the CSP framework, and can be easily generalised to the VCSP. For example, if $\mathcal{H}_{\mathcal{P}}$ is "tree-like", in various ways, then it can be shown that $\mathcal{P}$ is solvable in polynomial time via dynamic programming [ACP87, DP89, Fre90, Bod96, GLS00, KV00, GLS02, GGM$^+$05, CD05, GM06, AGG07, GMS07, Mar07, CJG08, Mar09a, Mar09b]. In fact, in the case of bounded-arity CSPs, the question of identifying all structural restrictions which guarantee tractability has been resolved completely: Grohe has shown that, under certain standard parameterised complexity theory assumptions,[4] the tractable class of bounded treewidth modulo homomorphic equivalence, identified in [DKV02], is the only tractable class, see [Gro07] for details. Note that structurally-restricted CSPs are also known as *uniform* CSP*s [KV07a].*

However, the complexity of finding an optimal solution to a valued constraint satisfaction problem will obviously also depend on the forms of valued constraints which are allowed in the problem [CCJK06]. Restricting the *forms* of the valued constraints which are allowed in the problem gives rise to so-called *language restrictions.*

Notation 2.2.1. A *valued constraint language* is simply a set of possible cost functions mapping D^k to $\overline{\mathbb{R}}$, for some fixed set D. A valued constraint language Γ over a two-element domain is called a *Boolean* valued constraint language.

Notation 2.2.2. We will denote by $\mathsf{VCSP}(\Gamma)$ the class of all VCSP instances where the cost functions of the valued constraints are all contained in the valued constraint language Γ.

Example 2.2.3. Let $D = \{0, 1\}$. We define two unary cost functions as follows:

$$\mu_2(x) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = 0, \\ 5 & \text{if } x = 1, \end{cases}$$

$$\mu_5(x) \stackrel{\text{def}}{=} \begin{cases} 4 & \text{if } x = 0, \\ 2 & \text{if } x = 1. \end{cases}$$

We also define six binary cost functions by the following table:

	ϕ_{12}	ϕ_{14}	ϕ_{23}	ϕ_{34}	ϕ_{35}	ϕ_{45}
00	3	0	0	9	3	4
01	2	4	1	7	5	3
10	3	2	0	8	4	2
11	1	5	0	1	4	1

The set $\Gamma = \{\mu_2, \mu_5, \phi_{12}, \phi_{14}, \phi_{23}, \phi_{34}, \phi_{35}, \phi_{45}\}$ is an example of a valued constraint language. We will now give an example of a $\mathsf{VCSP}(\Gamma)$ instance. Let $V =$

[4] Namely, that FPT$\neq$W[1] [FG06].

$\{x_1, x_2, x_3, x_4, x_5\}$ be a set of variables, and let $\mathcal{C}$ be a set of constraints, defined as:

$$\mathcal{C} = \{\langle\langle x_1, x_2\rangle, \phi_{12}\rangle, \langle\langle x_1, x_4\rangle, \phi_{14}\rangle, \langle\langle x_2, x_3\rangle, \phi_{23}\rangle, \\ \langle\langle x_3, x_4\rangle, \phi_{34}\rangle, \langle\langle x_3, x_5\rangle, \phi_{35}\rangle, \langle\langle x_4, x_5\rangle, \phi_{45}\rangle, \langle x_2, \mu_2\rangle, \langle x_5, \mu_5\rangle\}.$$

Then $\mathcal{P} = \langle V, D, \mathcal{C}\rangle$ is a $\mathsf{VCSP}(\Gamma)$ instance, illustrated in Figure 2.1.

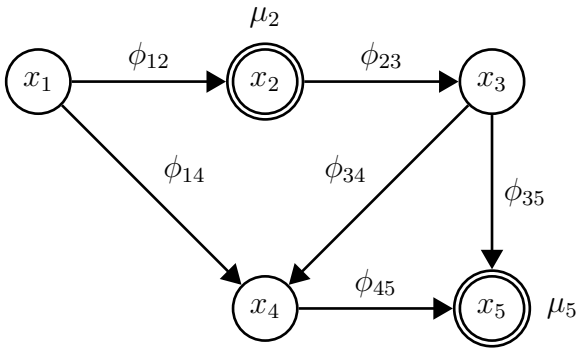

Figure 2.1: Instance $\mathcal{P}$ (Example 2.2.3).

Remark 2.2.4. As this is not a severe restriction, we will assume that every valued constraint language contains the equality relation and a constant function.

Notation 2.2.5. A valued constraint language Γ is called *tractable* if $\mathsf{VCSP}(\Gamma')$ can be solved in polynomial time for every finite subset $\Gamma' \subseteq \Gamma$, and Γ is called NP-hard (or *intractable*) if $\mathsf{VCSP}(\Gamma')$ is NP-hard for some finite subset $\Gamma' \subseteq \Gamma$.

Remark 2.2.6. A tractable Γ is sometimes called *locally tractable* as opposed to *globally tractable*, where the latter means that there is a uniform polynomial-time algorithm which solves $\mathsf{VCSP}(\Gamma)$.

Remark 2.2.7. Defining tractability in terms of finite subsets ensures that the tractability of a valued constraint language is independent of whether the cost functions are represented *explicitly* (via tables of values) or *implicitly* (via oracles). This is because for any finite $\Gamma' \subseteq \Gamma$, the algorithm for solving $\mathsf{VCSP}(\Gamma')$ can remember all the values of all cost functions in Γ'.

Our interest is in the effect of restricting the forms of cost functions allowed in valued constraint languages. In some cases, the restriction on the valued constraints may result in more tractable versions of the VCSP.

Example 2.2.8 ($\mathsf{VCSP}(\Gamma_{\mathsf{cut}})$)**.** Recall the valued constraint language Γ_{cut} from Example 2.1.15. Any instance of $\mathsf{VCSP}(\Gamma_{\mathsf{cut}})$ on variables $x_1, \ldots, x_n$ can be solved in $O(n^3)$ time by a standard (s,t)-Min-Cut algorithm [GT88], via the following reduction to (s,t)-Min-Cut: any unary constraint μ_c^0 (respectively μ_c^1) on x_i can be modelled by an edge of weight c from x_i to the target node (respectively, from the source node to the node x_i). Any $\lambda_w(x_i, x_j)$ constraint is modelled by an edge of weight w from x_i to x_j.

Example 2.2.9 (MIN-CUT)**.** Given a directed weighted graph $G = \langle V, E \rangle$ as in Example 2.1.15, a subset of vertices $C \subseteq V$ is called a *cut* if C is non-trivial, that is, $C \neq \emptyset$ and $C \neq V$. The weight of C is defined as in Example 2.1.15. The MIN-CUT problem, also known as the GLOBAL MIN-CUT problem, consists in finding a minimum-weight cut in G. Using the cubic-time algorithm for the (s,t)-MIN-CUT problem [GT88], one can easily construct an algorithm for the MIN-CUT problem of order $O(n^4)$, where $n = |V|$ is the number of vertices of G. For undirected graphs, Nagamochi and Ibaraki have described a more efficient algorithm, still based on network flows, which runs in cubic time [NI92]. Later, a simpler and purely combinatorial cubic-time algorithm which is not based on network flows has been discovered [SW97].

We have shown in [ŽJ09b] that MIN-CUT cannot be naturally described in the VCSP framework by any tractable valued constraint language over finite domains.

Next we give a list of several well-known problems which can be formulated in the language-restricted VCSP framework. We refer to the original papers which provide complexity *classification* results. In other words, they show under which language restrictions a given problem is tractable and under which restrictions it is intractable.

We start with an example which demonstrates that a valued constraint satisfaction problem involving only one single binary Boolean cost function can be NP-hard.

Example 2.2.10 ($\mathsf{VCSP}(\Gamma_{\mathsf{xor}})$)**.** Let Γ_{xor} be the Boolean valued constraint language which contains just the single binary cost function $\phi_{\mathsf{xor}} : D^2 \to \overline{\mathbb{R}}$ defined by

$$\phi_{\mathsf{xor}}(x,y) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{if } x \neq y. \end{cases}$$

If $D = \{0,1\}$, then the problem $\mathsf{VCSP}(\Gamma_{\mathsf{xor}})$ corresponds to the MAX-SAT problem for the exclusive-or predicate, which is known to be NP-hard [CKS01]. $\mathsf{VCSP}(\Gamma_{\mathsf{xor}})$ is also equivalent to the NP-complete MAX-CUT problem [GJ79]. For $|D| > 2$, the problem $\mathsf{VCSP}(\Gamma_{\mathsf{xor}})$ corresponds to the $|D|$-COLOURING problem, which is NP-complete. Therefore, Γ_{xor} is NP-hard.

Example 2.2.11 (SAT, MAX-SAT)**.** Consider a Boolean valued constraint language Γ. We can restrict Γ by allowing only cost functions with range $\{0,\infty\} \subseteq \overline{\mathbb{R}}$. This way we obtain the standard SATISFIABILITY problem. The complexity of the VCSP for such restricted languages Γ has been completely characterised and six tractable classes have been identified [Sch78].

Alternatively, if we restrict Γ by allowing only cost functions with range $\{0,1\} \subseteq \overline{\mathbb{R}}$, we obtain the MAX-SAT problem, in which the aim is to satisfy the maximum number of constraints. The complexity of this problem has been completely characterised and three tractable classes have been identified [CKS01].

Example 2.2.12 (CSP)**.** There has been a lot of research on the complexity of language-restricted CSPs. The first result in this line of research goes back to Schaefer, who obtained a complete classification of Boolean CSPs [Sch78], as mentioned in Example 2.2.11. Other known complexity classification results include CSPs with a

single binary symmetric relation (that is, graphs) [HN90] (see Example 2.2.22), CSPs with all unary relations (so-called *conservative* CSPs) [Bul03] (see Example 2.2.27), CSPs over a 3-element domain [Bul06], CSPs with a single binary relation without sources and sinks [BKN09], and a recent result classifying all CSPs with a single binary relation that belongs to a class of oriented trees called special triads [BKMN09].

Remark 2.2.13. It is known that the CSP is equivalent to the HOMOMORPHISM problem between relational structures and many other problems [FV98, HN04, CJ06, HN08]. Moreover, the CSP is equivalent to the CSP with a single binary relation [HN04].

Feder and Vardi have shown that the language-restricted CSP, also known as *nonuniform* CSP, is the biggest subclass of NP which can exhibit a *dichotomy* [FV98]. In other words, nonuniform CSPs form the biggest subclass of NP which might possibly contain only polynomial-time and NP-complete problems despite Ladner's Theorem, which shows existence of intermediate languages in NP provided P $\neq$ NP [Lad75]. Feder and Vardi conjectured that the nonuniform CSP does indeed exhibit a dichotomy [FV98]. Bulatov et al. have given an algebraic characterisation of this *Dichotomy Conjecture* [BKJ05]. Various equivalent formulations of the Dichotomy Conjecture can be found in a beautiful survey paper by Hell and Nešetřil [HN08], see also [NSZ09].

Feder and Vardi also showed that the nonuniform CSP is polynomial-time equivalent to a certain logic called MMSNP, that is, monotone monadic strict NP without inequality [FV98]. However, one of the reductions in [FV98] is a randomised polynomial-time reduction. More recently, Kun has proved a polynomial-time equivalence between the nonuniform CSP and MMSNP [Kun]. This equivalence has been refined by Kun and Nešetřil [KN08].

A variety of mathematical approaches to the nonuniform CSP have been suggested in the literature. The most advanced approaches use logic, combinatorics, universal algebra, and their combination [CJ06, Che06, KV07a, BKL08].

The logic programming language DATALOG can be used to define CSPs of bounded width, which can be solved by local consistency algorithms [Dec92, JCC98, DP99, KV00, AW09]. Recent results have established the power of the local consistency technique [LZ07, BK09]. In fact, all nonuniform CSPs that are known to be tractable can be solved either via local consistency techniques, or via the "few subpowers property" [IMM$^+$07, BIM$^+$] (which generalises Gaussian elimination and results in [FV98, JCC98, BD06, Dal06]), or via a combination of the two.

More on connections between logic and CONSTRAINT SATISFACTION can be found in [GKL$^+$07, HN08]. See also [CJ06] for a survey on the complexity of constraint languages, and [CKV08] for an overview of current research themes.

A recent result of Kun and Szegedy relates the Dichotomy Conjecture to continuous mathematics and techniques from PCPs (Probabilistically Checkable Proofs) [KS09].

Remark 2.2.14. The power of the local consistency technique, mentioned in Example 2.2.13, is also fully characterised for uniform CSPs [ABD07].

Example 2.2.15 (Boolean Min/Max-CSP, Min/Max-Ones)**.** Khanna et al. have obtained a complexity classification of Boolean Min-CSP and Max-CSP, and also Boolean Min-Ones and Max-Ones [KSTW01], see also [CKS01].

Example 2.2.16 (Boolean Min/Max-AW-CSP, Min/Max-AW-Ones)**.** Consider a generalisation of the Min-CSP and Max-CSP frameworks which allows arbitrary weights, that is, both positive and negative weights. In terms of cost functions, this means that each cost function can take on values 0 and c for some fixed (positive or negative) c depending on the cost function. Similarly for Min-Ones and Max-Ones.

Jonsson has generalised the results of Creignou et al. from Example 2.2.15 and has given a complexity classification of Boolean Min-AW-CSP and Max-AW-CSP, and also Min-AW-Ones and Max-AW-Ones [Jon00].

Example 2.2.17 (Boolean VCSP)**.** A complexity classification of Boolean VCSPs with arbitrary positive cost functions has been obtained by Cohen et al. [CCJK06].

Example 2.2.18 (Non-Boolean Max-CSP)**.** First results on the Max-CSP over arbitrary domains are due to Cohen et al. [CCJK05]. A complexity classification with respect to approximability of the three-valued Max-CSP is due to Jonsson et al. [JKK06].

Let Γ_{fix} be the language containing all unary relations of the form $x = d$ for some variable x and a domain value d (so-called *constant* or *fixed-value* constraints). A complexity classification of Max-CSPs with fixed-value constraints, that is, languages including Γ_{fix}, with respect to approximability, has been obtained by Deineko et al. [DJKK08].

See [Rag08] for recent results on the approximability and inapproximability of the Max-CSP.

Example 2.2.19 (Max-AW-CSP)**.** Jonsson and Krokhin have generalised results from Example 2.2.16 from Boolean domains to arbitrary domains, and obtained a complexity classification of Max-AW-CSPs, that is, Max-CSPs with arbitrary weights over arbitrary domains [JK07].

Example 2.2.20 (Max-Ones)**.** Jonsson et al. have generalised the result of Creignou et al. from Example 2.2.15, and have obtained a complexity classification with respect to approximability of the Max-Ones problem for maximal languages over domains of size up to 4 and of the Max-Ones problem with all permutation relations [JKN08].

Notation 2.2.21. Given two graphs (undirected or directed) G and H, we denote by $V(G)$ and $V(H)$ the set of vertices of G and H respectively. We denote by $E(G)$ and $E(H)$ the set of edges of G and H respectively. A mapping $f : V(G) \to V(H)$ is a *homomorphism* of G to H if f preserves edges, that is, $(u, v) \in E(G)$ implies $(f(u), f(v)) \in E(H)$.

Example 2.2.22 (Graph Homomorphism)**.** The Graph Homomorphism problem asks whether an input graph G admits a homomorphism to a fixed graph H. This problem is also known as H-Colouring [HN04].

H-Colouring is equivalent to $\mathsf{VCSP}(\Gamma_{\mathsf{e}})$, where Γ_{e} denotes the language containing a single binary symmetric relation representing the edges of H ("e" for edge). A complexity classification of the H-Colouring problem has been obtained by Hell and Nešetřil [HN90]: H-Colouring is tractable if, and only if, H contains a loop or H is bipartite; otherwise H-Colouring is NP-complete. Bulatov has provided an algebraic proof of this result [Bul05].

Example 2.2.23 (Graph List Homomorphism)**.** The Graph List Homomorphism problem for H asks whether an input graph G with lists $L_u \subseteq V(H)$, $u \in V(G)$ admits a homomorphism f to H such that $f(u) \in L_u$ for each $u \in V(G)$.

Let Γ_{cons} consist of all unary relations ("cons" for conservative). Then $\mathsf{VCSP}(\Gamma_{\mathsf{e}} \cup \Gamma_{\mathsf{cons}})$, where Γ_{e} is from Example 2.2.22, is equivalent to the Graph List Homomorphism problem. A complexity classification of this problem is due to Feder et al. [FHH03].

Example 2.2.24 (Graph Min-Cost Homomorphism)**.** For two graphs G and H, consider real nonnegative costs $c_v(u)$ for $u \in V(G)$ and $v \in V(H)$. The cost of an homomorphism f of G to H is defined to be $\sum_{u \in V(G)} c_{f(u)}(u)$. For a fixed H, the Graph Min-Cost Homomorphism problem asks to find a homomorphism of G to H with minimum cost.

Graph Min-Cost Homomorphism is equivalent to $\mathsf{VCSP}(\Gamma_{\mathsf{e}} \cup \Gamma_{\mathsf{scons}})$, where Γ_{e} is from Example 2.2.22, and Γ_{scons} consists of all unary cost functions ("scons" for soft conservative). A complexity classification of this problem is due to Gutin et al. [GHRY08].

Remark 2.2.25. We remark that structurally-restricted variants of the Graph Homomorphism problems from Examples 2.2.22, 2.2.23 and 2.2.24 have also been studied, see [Gro07, FJ07].

Example 2.2.26 (Digraph Homomorphism)**.** The Digraph Homomorphism problem is an analogue of Graph Homomorphism from Example 2.2.22 for directed graphs.

Let Γ_{a} denote the language containing a single binary relation ("a" for arc). For any fixed Γ_{a}, $\mathsf{VCSP}(\Gamma_{\mathsf{a}})$ is polynomial-time equivalent to the Digraph Homomorphism problem for the graph whose edge relation is given by the binary relation from Γ_{a} [HN04]. A complexity classification of the Digraph Homomorphism problem for semicomplete digraphs has been obtained in [BJHM88].

Example 2.2.27 (Digraph List Homomorphism)**.** The Digraph List Homomorphism problem is an analogue of Graph List Homomorphism from Example 2.2.23 for directed graphs.

$\mathsf{VCSP}(\Gamma_{\mathsf{a}} \cup \Gamma_{\mathsf{cons}})$, where Γ_{a} is from Example 2.2.26, and Γ_{cons} is from Example 2.2.23, is equivalent to the Digraph List Homomorphism problem. Bulatov has obtained a complexity classification of this problem [Bul03].

Example 2.2.28 (Digraph Min-Cost Homomorphism)**.** The Digraph Min-Cost Homomorphism Problem is an analogue of Graph Min-Cost Homomorphism problem from Example 2.2.24 for directed graphs.

$\mathsf{VCSP}(\Gamma_{\mathsf{a}} \cup \Gamma_{\mathsf{sconst}})$, where Γ_{a} is from Example 2.2.26 and Γ_{sconst} is from Example 2.2.24, is equivalent to the Digraph Min-Cost Homomorphism problem. This problem is also equivalent to the Level of Repair Analysis problem [GRYT06], see also [GK08]. Complexity classification results have been obtained for semicomplete digraphs [GRY06], semicomplete bipartite digraphs [GGY08], semicomplete multipartite digraphs [GRY08b], semicomplete digraphs with possible loops [GK09], locally semicomplete and quasi-transitive digraphs [GGK+09], locally in-semicomplete digraphs [GKKR08], reflexive digraphs [GHKR08] (generalising reflexive multipartite tournaments [GK07]), and oriented cycles [GRY08a].

Example 2.2.29 (Max-Sol)**.** The Maximum Solution problem is equivalent to the VCSP over the language consisting of all relations and unary cost functions with the following cost functions: $\mu(d) = wd$ for any domain value d and some fixed $w \in \mathbb{N}$ [JN08]. Jonsson et al. have studied the Max-Sol problem over graphs, that is, a language consisting of a single symmetric binary relation (this is a restriction of Graph Min-Cost Homomorphism from Example 2.2.22, but a generalisation of both Graph List Homomorphism from Example 2.2.23 and Max-Ones from Example 2.2.20) [JNT07].

Example 2.2.30 (#CSP)**.** The complexity of counting solutions to various combinatorial problems was first considered in [Val79]. In the Counting Constraint Satisfaction problem, #CSP, the goal is to find the number of solutions. The general framework is similar to the VCSP: instead of minimising the sum of cost functions (over all possible assignments of values to variables), the objective is to compute the sum (again, over all possible assignments of values to variables) of the product of all $\{0,1\}$-valued cost functions. Formally, for a #CSP instance $\mathcal{P} = \langle V, D, \mathcal{C} \rangle$, the goal is to compute

$$\mathsf{Eval}(\mathcal{P}) \stackrel{\text{def}}{=} \sum_{s \in \mathcal{A}} \prod_{\langle \mathbf{x}, \phi \rangle \in \mathcal{C}} \phi(\mathbf{x}).$$

(Recall from Definition 2.1.7 that $\mathcal{A}$ denotes the set of all assignments of values to the variables.)

Below is a list of results on complexity classifications of #CSPs.

- Creignou and Hermann have classified Boolean #CSPs with an arbitrary number of cost functions of arbitrary fixed arities [CH96], see also [CKS01].
- Dyer and Greenhill have obtained a complexity classification of #CSPs with only one symmetric cost function of arity 2 over an arbitrary finite domain [DG00]. This problem is also known as $\#H$-Colouring.
- Dyer et al. have obtained a complexity classification of #CSPs with only one non-symmetric acyclic cost function of arity 2 over an arbitrary finite domain [DGP07].
- Dyer et al. have obtained a complexity classification (with respect to approximability) of Boolean #CSPs with an arbitrary number of cost functions of arbitrary fixed arities [DGJ09a].

- Dyer et al. have obtained a complexity classification (with respect to approximability) of Boolean #CSPs with bounded degree [DGJR09].

- Building on work of Bulatov and Grohe [BG05] and Bulatov and Dalmau [BD07], Bulatov has obtained a complexity classification of #CSPs with an arbitrary number of cost functions of arbitrary fixed arities over an arbitrary finite domain [Bul08]. However, the answer is not completely satisfactory as Bulatov's results show that there is a dichotomy, but it is not known whether recognising the tractable cases is even decidable [Bul08].

Example 2.2.31 (Partition Function)**.** The Counting Constraint Satisfaction problem, #CSP, from Example 2.2.30 can be generalised from $\{0, 1\}$-valued cost functions to arbitrary cost functions. This is known as the Partition Function problem, the Weighted #CSP problem or just as #CSP. Recall that the goal is to compute the sum (over all possible assignments of values to variables) of the product of all cost functions in a given instance. Note that in this case the resulting number does not correspond to the number of solutions anymore as the concept of number of solutions does not make any sense. Below is a list of complexity classification results on #CSPs for general cost functions, that is, cost functions that are not necessarily $\{0, 1\}$-valued as in Example 2.2.30:

- Bulatov and Grohe have obtained a complexity classification of #CSPs with only one symmetric cost function of arity 2 taking non-negative real values [BG05].

- Goldberg et al. have obtained a complexity classification of #CSPs with only one symmetric cost function of arity 2 taking both positive and negative real values [GGJT09].

- Cai et al. have obtained a complexity classification of #CSPs with only one symmetric cost function of arity 2 taking complex values [CCL09].

- Dyer et al. have obtained a complexity classification of #CSPs with only one symmetric cost function of an arbitrary fixed arity taking non-negative rational values [DGJ08]. This problem is also known as the Hypergraph Partition Function problem.

- Dyer et al. have obtained a complexity classification of Boolean #CSPs with arbitrarily many cost functions of arbitrary fixed arities taking non-negative real values [DGJ09b].

- Bulatov et al. have extended the previous result to a complexity classification of Boolean #CSPs with arbitrarily many cost functions of arbitrary fixed arities taking both positive and negative rational values [BDG$^+$09].

- Cai et al. have extended this result to a complexity classification of Boolean #CSPs with arbitrarily many cost functions of arbitrary fixed arities taking complex values [CLX08, CLX09].

Remark 2.2.32. A dichotomy result for structurally-restricted #CSPs with arbitrarily many cost functions of arbitrary (but fixed) arities over arbitrary domains is due to Dalmau and Jonsson [DJ04].

Surprisingly, not much research has been done on the combination of the two mentioned restrictions (that is, language and structural restrictions), which would result in finding *hybrid* reasons for tractability [Coh03, CJS08, SJ08], see also [ŽJ09b].

2.3 Expressibility

In this section, we introduce the concept of expressibility for valued constraints. We also give some illustrative examples.

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 all variables, then their values are constrained *implicitly* in the same way, due to the combined effect of the valued constraints. Recall Example 1.8, where variables x_1 and x_4, even though not constrained explicitly, are constrained implicitly by the equality relation. This motivates the concept of *expressibility* for cost functions, which is defined as follows:

Definition 2.3.1 (Expressibility)**.** For any VCSP instance $\mathcal{P} = \langle V, D, \mathcal{C}\rangle$, and any tuple $\mathbf{x}$ of m variables of $\mathcal{P}$, the *projection* of $\mathcal{P}$ onto $\mathbf{x}$, denoted $\pi_{\mathbf{x}}(\mathcal{P})$, is the m-ary cost function defined as

$$\pi_{\mathbf{x}}(\mathcal{P})(\mathbf{y}) \stackrel{\text{def}}{=} \min_{s\in\mathcal{A}}\{Cost_{\mathcal{P}}(s) \mid s(\mathbf{x}) = \mathbf{y}\},$$

where $\mathcal{A}$ denotes the set of all assignments (Definition 2.1.7). We say that a cost function ϕ is *expressible* over a valued constraint language Γ if there exists an instance $\mathcal{P}$ of $\mathsf{VCSP}(\Gamma)$ and a tuple $\mathbf{v}$ of variables of $\mathcal{P}$ such that $\pi_{\mathbf{v}}(\mathcal{P}) = \phi$. The variables of $\mathcal{P}$ not from $\mathbf{v}$ are called *extra* (or *hidden*) variables. We call the pair $\langle\mathcal{P}, \mathbf{v}\rangle$ a *gadget* for expressing ϕ over Γ.

Remark 2.3.2. The notion of expressibility is also known as *implementation* [CKS01].

Remark 2.3.3. The notion of expressibility for crisp cost functions (=relations) corresponds to expressibility using conjunction and existential quantification (*primitive positive formulas*) [BKJ05]. Hence relations expressible over a crisp constraint language Γ are also known as pp-definable over Γ [Che06].

The following result states that adding a cost function expressible over Γ to Γ does not change the complexity of $\mathsf{VCSP}(\Gamma)$.

Theorem 2.3.4. *For any valued constraint language Γ and any ϕ expressible over Γ, $\mathsf{VCSP}(\Gamma)$ and $\mathsf{VCSP}(\Gamma \cup \{\phi\})$ are log-space equivalent.*

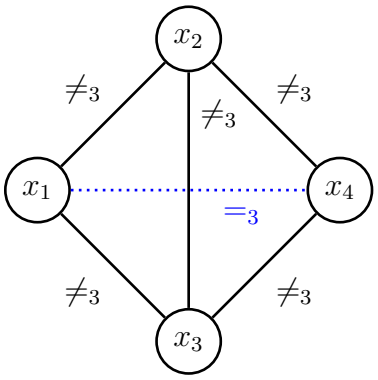

Figure 2.2: The gadget expressing $=_3$ over $\{\neq_3\}$ for $|D| = 3$ (Example 2.3.7).

Remark 2.3.5. Note that the original version of this result from [CCJK06] shows only polynomial-time equivalence. However, this result can be extended due to Reingold's result that (s,t)-Connectivity in undirected graphs can be solved in logarithmic space [Rei08].

Example 2.3.6. By Theorem 2.3.4 and Example 2.2.10, in order to show that Γ is an NP-hard valued constraint language it is sufficient to show that ϕ_{xor} is expressible over Γ.

Example 2.3.7. Let D be a finite set of size d. Consider a valued constraint language $\Gamma = \{\neq_d\}$ over D which consists of a binary disequality relation, $\neq_d$, given by

$$\neq_d \stackrel{\text{def}}{=} \{\langle a, b\rangle \in D^2 \mid a \neq b\}.$$

Note that $\neq_d$ is a "hard variant" of ϕ_{xor} from Example 2.2.10. Consider an instance $\mathcal{P} = \{V, D, \mathcal{C}\}$ of $\mathsf{VCSP}(\Gamma)$, where $V = \{x_1, \ldots, x_{n+1}\}$, $n = d$, and

$$\mathcal{C} = \{\langle\langle x_i, x_j\rangle, \neq_d\rangle \mid i \neq j \in \{1, \ldots, n\}\} \cup \{\langle\langle x_i, x_{n+1}\rangle, \neq_d\rangle \mid i \in \{2, \ldots, n\}\}.$$

In order to satisfy all constraints from $\mathcal{C}$, variables $x_1, \ldots, x_n$ have to be assigned different values. Moreover, the value of the variable x_{n+1} has to be different from the values of the variables $x_2, \ldots, x_n$. Hence, the only remaining value that can be assigned to the variable x_{n+1} is the value which is assigned to the variable x_1. Therefore, every solution s to $\mathcal{P}$ with minimum total cost (in this case zero) satisfies $s(x_1) = s(x_{n+1})$. Therefore, $\langle \mathcal{P}, \{x_1, x_{n+1}\}\rangle$ is a gadget for the equality relation, $=_d$, given by

$$=_d \stackrel{\text{def}}{=} \{\langle a, b\rangle \in D^2 \mid a = b\}.$$

In other words, the equality relation can be expressed using the disequality relation. An example of this construction for $|D| = 3$ is shown in Figure 2.2.

Example 2.3.8. Consider the VCSP instance $\mathcal{P}$ from Example 2.2.3. The projection of $\mathcal{P}$ onto $\langle x_2, x_4\rangle$, denoted by $\pi(\mathcal{P})_{\langle x_2, x_4\rangle}$, is a binary cost function defined by

minimising over the remaining variables. The following table, which enumerates all assignments s in which x_2 and x_4 are both assigned 0, together with the cost of these assignments, shows that $\pi(\mathcal{P})_{\langle x_2,x_4\rangle}(0,0) = 21$.

x_2	x_4	x_1	x_3	x_5	$Cost_{\mathcal{P}}(s)$
0	0	0	0	0	23
0	0	0	0	1	22
0	0	0	1	0	24
0	0	**0**	**1**	**1**	**21**
0	0	1	0	0	25
0	0	1	0	1	24
0	0	1	1	0	26
0	0	1	1	1	23

Similarly, it is straightforward to check that

$$\pi(\mathcal{P})_{\langle x_2,x_4\rangle}(x,y) = \begin{cases} 21 & \text{if } x = 0 \text{ and } y = 0, \\ 16 & \text{if } x = 0 \text{ and } y = 1, \\ 24 & \text{if } x = 1 \text{ and } y = 0, \\ 19 & \text{if } x = 1 \text{ and } y = 1. \end{cases}$$

Hence this cost function can be expressed over the valued constraint language Γ defined in Example 2.2.3.

Example 2.3.9. Consider a ternary finite-valued cost function ϕ over $D = \{0,1,2\}$ defined as $\phi = (\#0)^2$, that is, the square of the number of zeros in the input. We will construct a gadget for expressing ϕ using only binary crisp cost functions and finite-valued unary cost functions.

Define three binary crisp cost functions as follows:

$$\phi_0(x,y) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } x = 0 \text{ and } y = 1, \\ \infty & \text{if } x = 0 \text{ and } y = 2, \\ 0 & \text{otherwise,} \end{cases}$$

$$\phi_1(x,y) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } x = 0 \text{ and } y = 1, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\phi_2(x,y) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } x = 0 \text{ and } y = 2, \\ 0 & \text{otherwise.} \end{cases}$$

For $c \in \{1,3,5\}$, let μ_c be a unary finite-valued cost function defined as

$$\mu_c(x) \stackrel{\text{def}}{=} \begin{cases} c & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases}$$

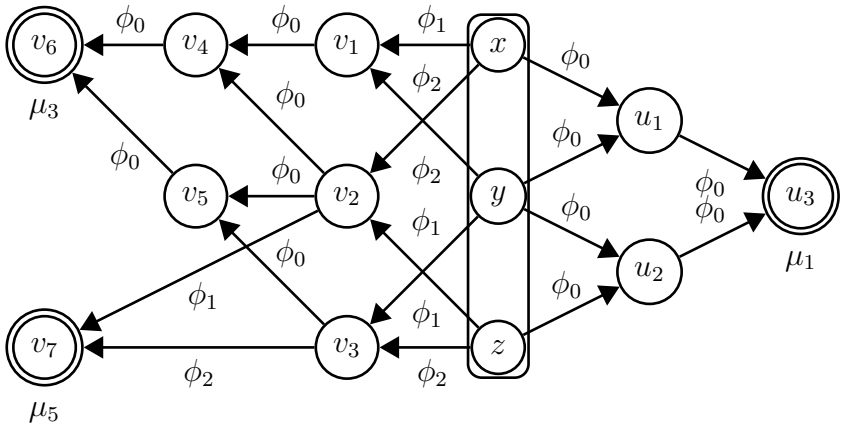

Figure 2.3: The gadget expressing $\phi = (\#0)^2$ (Example 2.3.9).

Let $\mathcal{P} = \langle V, D, \mathcal{C} \rangle$ where $V = \{x, y, z, u_1, u_2, u_3, v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ and the set of constraints $\mathcal{C}$ is shown in Figure 2.3.

We claim that $\langle \mathcal{P}, \langle x, y, z \rangle \rangle$ is a gadget for expressing ϕ.

If all x, y and z are non-zero, then there is an assignment of the other variables with values one and two such that the total cost is 0.

If any of x, y, z is zero, then in any minimum-cost assignment either u_1 or u_2 is assigned zero, and for the same reason u_3 is assigned zero.

If at least two of x, y, z are zero, then in any minimum-cost assignemnt at least one of the variables v_1, v_2, v_3 is assigned zero, and consequently, at least one of v_4, v_5 is assigned zero, and hence v_6 is assigned 0.

If all x, y and z are zero, then both v_2 and v_3 are assigned zero, and consequently, v_7 is assigned zero.

Note that a similar gadget can be constructed for bigger domains.

Two more examples can be found in Chapter 1 (Example 1.10 and 1.9).

2.4 Algebraic properties

In this section, we describe some algebraic techniques that have been developed for valued constraints and show how they can be used to investigate expressibility.

Note that adding a finite constant to any cost function does not alter the relative costs.

Definition 2.4.1 (Expressive Power)**.** For any valued constraint language Γ with costs in $\overline{\mathbb{R}}$, 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 finite constant $c \in \mathbb{R}$.

A number of algebraic techniques to determine the expressive power of a given valued constraint language have been developed in the literature. 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 the set D in a standard way, as follows. For any function $f : D^k \to D$, and any collection of tuples $t_1, \ldots, t_k \in D^m$, define $f(t_1, \ldots, t_k) \in D^m$ to be the tuple $\langle f(t_1[1], \ldots, t_k[1]), \ldots, f(t_1[m], \ldots, t_k[m])\rangle$. For instance, $f(\langle u_1, \ldots, u_k\rangle, \langle v_1, \ldots, v_k\rangle) = \langle f(u_1, v_1), \ldots, f(u_k, v_k)\rangle$.

Definition 2.4.2 (Polymorphism [DW02])**.** 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 $f(t_1, \ldots, t_k) \in R$ for all choices of $t_1, \ldots, t_k \in R$.

Notation 2.4.3. We denote by $\pi_i : D^k \to D$ the *projection* operation which returns its i-th argument, that is, $\pi_i(x_1, \ldots, x_n) = x_i$.

Observation 2.4.4. It is easy to observe that any projection π_i is a polymorphism of all relations.

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.

Notation 2.4.5. The set of all polymorphisms of Γ will be denoted $\mathsf{Pol}(\Gamma)$.

It follows from the results of Geiger and Bodnarčuk et al. that the expressive power of a crisp constraint language is fully characterised by its polymorphisms:

Theorem 2.4.6 ([Gei68, BKKR69, Jea98])**.** *For any crisp constraint language* Γ *over a finite set*

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

Hence, a relation R is expressible over a crisp constraint language Γ if, and only if, it has all the polymorphisms of Γ. See [JCG97] for more on the connection between crisp constraint languages on the one hand, and universal algebra on the other hand. We will discuss this in more depth in Chapter 3.

Remark 2.4.7. It is known that crisp constraint languages which have only trivial polymorphisms are NP-hard [BKJ05], where trivial means projections and semi-projections. (Note that this is a powerful technique to show NP-hardness results as one does not need to build a reduction from some known NP-hard problem.) The famous *Dichotomy Conjecture*, mentioned in Remark 2.2.13, can be equivalently stated as follows: if a crisp constraint language Γ has a non-trivial polymorphism, then $\mathsf{CSP}(\Gamma)$ is polynomial-time solvable. In fact, there is only one remaining type of polymorphism for which we currently do not know whether it leads to polynomial or NP-hard problems, and those are *Taylor* polymorphisms [HN08]. Answering this question would resolve the Dichotomy Conjecture. Maróti and McKenzie have shown that Taylor polymorphism are equivalent, with respect to the question of tractability, to so-called *weak near-unanimity* polymorphisms [MM08].

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

Definition 2.4.8 (Feasibility Polymorphism [CCJ06])**.** The *feasibility polymorphisms* of a valued constraint language Γ are the polymorphisms of the corresponding crisp feasibility cost functions, that is,

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

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

$$\begin{array}{lllll} t_1 & t_1[1] & t_1[2] & \dots & t_1[m] \\ t_2 & t_2[1] & t_2[2] & \dots & t_2[m] \\ \vdots & & & \vdots & \\ t_k & t_k[1] & t_k[2] & \dots & t_k[m] \end{array} \quad \xrightarrow{\phi} \quad \left.\begin{array}{l} \phi(t_1) \\ \phi(t_2) \\ \vdots \\ \phi(t_k) \end{array}\right\} \sum_{i=1}^{k} \phi(t_i)$$

$$\begin{array}{lllll} t'_1 = f_1(t_1,\dots,t_k) & t'_1[1] & t'_1[2] & \dots & t'_1[m] \\ t'_2 = f_2(t_1,\dots,t_k) & t'_2[1] & t'_2[2] & \dots & t'_2[m] \\ \vdots & & & \vdots & \\ t'_n = f_n(t_1,\dots,t_k) & t'_n[1] & t'_n[2] & \dots & t'_n[m] \end{array} \quad \xrightarrow{\phi} \quad \left.\begin{array}{l} \phi(t'_1) \\ \phi(t'_2) \\ \vdots \\ \phi(t'_n) \end{array}\right\} \begin{array}{l} \mathsf{I}\vee \\ \displaystyle\sum_{i=1}^{n} r_i \phi(t'_i) \end{array}$$

Figure 2.4: Definition of a fractional polymorphism $\mathcal{F} = \{\langle r_1, f_1\rangle, \dots, \langle r_n, f_n\rangle\}$.

Definition 2.4.9 (Weighted Mapping [CCJ06])**.** A k-ary *weighted mapping* $\mathcal{F}$ 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 positive rational number such that $\sum_{i=1}^{n} r_i = k$ and each f_i is a distinct function from D^k to D.

Definition 2.4.10 (Fractional Polymorphism [CCJ06])**.** For any m-ary cost function ϕ, we say that a k-ary weighted mapping $\mathcal{F}$ is a k-ary *fractional polymorphism* of ϕ if, for all $t_1, \dots, t_k \in D^m$,

$$\sum_{i=1}^{k} \phi(t_i) \;\geq\; \sum_{i=1}^{n} r_i \phi(f_i(t_1, \dots, t_k)). \tag{2.1}$$

See Figure 2.4 for an illustration of Definition 2.4.10. (We call such a table a *tableau*.)

Definition 2.4.11 (Multimorphism [CCJK06])**.** A k-ary weighted fractional polymorphism whose weights are all natural numbers is called a *multimorphism*.

See Figure 2.5 for an illustration of Definition 2.4.11.

Remark 2.4.12. Given a multimorphism $\mathcal{F} = \langle\langle r_1, f_1\rangle, \dots, \langle r_n, f_n\rangle\rangle$, where each r_i is a natural number, the sum $\sum_{i=1}^{n} r_i$ is equal to k, and each f_i is a function from D^k to D, $\mathcal{F}$ can be seen as a mapping from D^k to D^k.

$$
\begin{array}{lllll}
t_1 & t_1[1] & t_1[2] & \dots & t_1[m] \\
t_2 & t_2[1] & t_2[2] & \dots & t_2[m] \\
\vdots & & & \vdots & \\
t_k & t_k[1] & t_k[2] & \dots & t_k[m]
\end{array}
\quad \xrightarrow{\phi} \quad
\left.
\begin{array}{l}
\phi(t_1) \\ \phi(t_2) \\ \vdots \\ \phi(t_k)
\end{array}
\right\} \sum_{i=1}^{k} \phi(t_i)
$$

$$
\begin{array}{lllll}
t'_1 = f_1(t_1,\dots,t_k) & t'_1[1] & t'_1[2] & \dots & t'_1[m] \\
t'_2 = f_2(t_1,\dots,t_k) & t'_2[1] & t'_2[2] & \dots & t'_2[m] \\
\vdots & & & \vdots & \\
t'_k = f_n(t_1,\dots,t_k) & t'_k[1] & t'_k[2] & \dots & t'_k[m]
\end{array}
\quad \xrightarrow{\phi} \quad
\left.
\begin{array}{l}
\phi(t'_1) \\ \phi(t'_2) \\ \vdots \\ \phi(t'_k)
\end{array}
\right\} \sum_{i=1}^{k} \phi(t'_i)
$$

Figure 2.5: Definition of a multimorphism $\mathcal{F} = \langle f_1, \dots, f_k \rangle$.

Remark 2.4.13. A cost function $\phi : D^m \to \overline{\mathbb{R}}$ has a multimorphism $\mathcal{F}$ if ϕ satisfies a set of linear inequalities. Hence from the geometrical point of view, for each possible fixed arity m, $\mathcal{F}$ corresponds to a hyperplane in a space of dimension $|D|^m$ [Sch86].

Remark 2.4.14. An equivalent definition of a fractional polymorphism is given in [CCJ06]: all weights have to be natural numbers; however, there is no restriction on the sum of all weights. Such a fractional polymorphism is a multimorphism if the sum of all weights is equal to k.

Notation 2.4.15. For any set of cost functions Γ, we denote by $\mathsf{fPol}(\Gamma)$ the set of all $\mathcal{F}$ such that $\mathcal{F}$ is a fractional polymorphism of every cost function in Γ. Similarly, $\mathsf{Mul}(\Gamma)$ denotes the set of multimorphisms of all cost functions from Γ.

Observation 2.4.16. It is a simple consequence of the definitions that if $\mathcal{F} = \{\langle r_1, f_1 \rangle, \dots, \langle r_n, f_n \rangle\}$ is a fractional polymorphism of ϕ, then $\{f_i\}_{1 \le i \le n}$ are feasibility polymorphisms of ϕ. On the other hand, if $\{f_i\}_{1 \le i \le k}$ are feasibility polymorphisms of ϕ, then $\langle f_1, \dots, f_k \rangle$ is not necessarily a multimorphism, and therefore not necessarily a fractional polymorphism, of ϕ. However, in the case of crisp cost functions the relationship is tighter. If $\{f_i\}_{1 \le i \le n}$ are feasibility polymorphisms of a relation R, then any weighted mapping $\{\langle r_1, f_1 \rangle, \dots, \langle r_n, f_n \rangle\}$ is a fractional polymorphism of the corresponding crisp cost function ϕ_R.

Observation 2.4.17 ([CCJK06])**.** If Γ is a tractable valued constraint language, then the set of relations $\{\mathsf{Feas}(\phi) \mid \phi \in \Gamma\}$ must be a tractable crisp constraint language.

Observation 2.4.18. A *multi-projection* is a mapping from D^k to D^k that only permutes the set of its arguments. It follows from Definition 2.4.11 that every multi-projection is a multimorphism of all cost functions.

It has been shown in [CCJ06] that the feasibility polymorphisms and fractional polymorphisms of a valued constraint language effectively determine its expressive power. One consequence of this result is the following theorem:

Theorem 2.4.19 ([CCJ06])**.** *If Γ is a valued constraint language with costs in $\overline{\mathbb{R}}$, Γ contains a constant function and is closed under scaling and the feasibility operator,*[5] *then*

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

Proof. This theorem is established in [CCJ06] for cost functions taking positive rational values, but the proofs are easily extended to cost functions taking arbitrary real values. A slightly weaker notion of expressibility is also used in [CCJ06], which does not allow arbitrary scaling, but again the proof is easily extended. □

Corollary 2.4.20. *For all suitable valued constraint languages Γ, a cost function ϕ is expressible over Γ if, and only if, it has all the feasibility polymorphisms and fractional polymorphisms of Γ.*

We will provide more details on Theorem 2.4.19 in Chapter 3. The next result shows that adding cost functions which are expressible over a given valued constraint language to the language does not change the complexity of the language.

Theorem 2.4.21 ([CCJ06])**.** *A valued constraint language Γ is tractable if, and only if, $\langle\Gamma \cup \mathsf{Feas}(\Gamma)\rangle$ is tractable.*

We finish this section with a simple example of the algebraic technique. We have shown in Example 2.3.7 that the disequality relation can express the equality relation. We now investigate the converse question.

Example 2.4.22. Consider a constraint language $\Gamma = \{=_d\}$ over D, $|D| = d$, which consists of the binary equality relation $=_d$ from Example 2.3.7. Consider k arbitrary 2-tuples $t_1, \ldots, t_k$ over D and an arbitrary function $f : D^k \to D$. If $t_i \in \{=_d\}$ for every $i = 1, \ldots, k$, then $f(t_1[1], \ldots, t_k[1]) = f(t_1[2], \ldots, t_k[2])$ and therefore $f(t_1, \ldots, t_k) \in \{=_d\}$. It follows that every function is a polymorphism of $=_d$. Obviously, not every function is a polymorphism of $\neq_d$: a simple counterexample is a constant function. We have shown that $\mathsf{Pol}(\{=_d\}) \not\subseteq \mathsf{Pol}(\{\neq_d\})$ and therefore $\neq_d$ is not expressible over $\{=_d\}$ by Theorem 2.4.6. This is almost obvious, but this simple example illustrates the use of the algebraic approach.

2.5 Submodularity

In this section, we define submodular functions, and the submodular function minimisation problem.

For any finite set V, a real-valued function f defined on subsets of V is called a *set function.*

Definition 2.5.1. A set function $f : 2^V \to \mathbb{R}$ is called *submodular* if for all $S, T \subseteq V$,

$$f(S \cap T) + f(S \cup T) \;\leq\; f(S) + f(T).$$

[5] That is, for every $\phi \in \Gamma$ and every $c \in \mathbb{R}_+$, $c\phi \in \Gamma$ and $\mathsf{Feas}(\phi) \in \Gamma$.

Remark 2.5.2. An equivalent definition of submodularity is the property of *decreasing marginal values:* for any $A \subseteq B \subseteq V$ and $x \in V \setminus B$, $f(B \cup \{x\}) - f(B) \leq f(A \cup \{x\}) - f(A)$. This can be deduced from the first definition by substituting $S = A \cup \{x\}$ and $T = B$; the reverse implication also holds [Sch03].

Submodular functions are a key concept in operational research and combinatorial optimisation [NW88, Nar97, Top98, Sch03, Fuj05, KV07b, Iwa08]. Examples include cuts in graphs [GW95, Que98], matroid rank functions [Edm70], set covering problems [Fei98] and entropy functions. Submodular functions are often considered to be a discrete analogue of convex functions [Lov83].

Both minimising and maximising submodular functions, possibly under some additional conditions, have been considered extensively in the literature. Most scenarios use the so-called *oracle value model*: for any set S, an algorithm can query an oracle to find the value of $f(S)$.

Submodular function maximisation is easily shown to be NP-hard [Sch03] since it generalises many standard NP-hard problems such as the Max-Cut problem, see also [FMV07]. In contrast, the Submodular Function Minimisation problem, SFM, which consists in *minimising* a submodular function, can be solved efficiently with only polynomially many oracle calls.

Notation 2.5.3. Let $\mathcal{B}$ be an algorithm for the minimisation problem of f in the oracle value model. $\mathcal{B}$ is called *polynomial* if it runs in polynomial time. A polynomial algorithm $\mathcal{B}$ is called *strongly polynomial* if the running time does not depend on $M = \max f$. In other words, the number of elementary arithmetic operations and other operations is bounded by a polynomial in the size of the input. A polynomial algorithm $\mathcal{B}$ which does depend on M is called *weakly polynomial*. A polynomial algorithm $\mathcal{B}$ is called *combinatorial* if it does not employ the ellipsoid method. Finally, a combinatorial algorithm $\mathcal{B}$ is called *fully combinatorial* if it uses only oracle calls, additions, subtractions and comparisons, but not multiplications and divisions, as fundamental operations.

The first polynomial algorithm for the SFM problem is due to Grötschel, Lovász and Schrijver [GLS81]. A strongly polynomial algorithm has been described in [GLS88]. These algorithms employ the ellipsoid method.

Based on the work of Cunningham [Cun84, Cun85], several combinatorial algorithms have been obtained in the last decade [Sch00, IFF01, Iwa02, Iwa03, FI03, Orl09, IO09]. The first fully combinatorial algorithm for the SFM has been described in [Iwa02], improved in [Iwa03], and recently improved again (without using the scaling method) in [IO09].

The time complexity of the fastest known general strongly polynomial algorithm for the SFM is $O(n^6 + n^5 L)$, where n is the number of variables and L is the time required to evaluate the function [Orl09].

Remark 2.5.4. The minimisation of submodular functions on sets is equivalent to the minimisation of submodular functions on distributive lattices [Sch03]. Krokhin and Larose have also studied the more general problem of minimising submodular functions on non-distributive lattices [KL08].

An important and well-studied sub-problem of SFM is the minimisation of submodular functions of bounded arity (SFM_b), also known as *locally-defined* submodular functions [Coo08b], or submodular functions with *succinct representation* [FMV07]. In this scenario the submodular function to be minimised is defined as the sum of a collection of functions which each depend only on a bounded number of variables. Locally defined optimisation problems of this kind occur in a wide variety of contexts:

- In the context of Pseudo-Boolean Optimisation, such problems involve the minimisation of Boolean polynomials of bounded *degree* [BH02, CH].
- In the context of computer vision, such problems are often formulated as Gibbs Energy Minimisation problems [GG84] or Markov Random Fields (also known as Conditional Random Fields) [Lau96, WJ08].

We now define the concept of submodularity for valued constraints.

Recall that L is a *lattice* if L is a partially ordered set in which every pair of elements has a unique supremum and a unique infimum. For a finite lattice L and a pair of elements (a, b), we will denote the unique supremum of a and b by $\textsc{Max}(a, b)$ (or $a \vee b$), and the unique infimum of a and b by $\textsc{Min}(a, b)$ (or $a \wedge b$).

Definition 2.5.5 (Submodularity)**.** Let D be a finite lattice-ordered set. A cost function $\phi : D^m \to \overline{\mathbb{R}}$ is *submodular* if $\langle \textsc{Min}, \textsc{Max} \rangle \in \mathsf{Mul}(\{\phi\})$, that is, for all m-tuples u, v,

$$\phi(\textsc{Min}(u, v)) + \phi(\textsc{Max}(u, v)) \leq \phi(u) + \phi(v). \tag{2.2}$$

Note that the SFM_b is equivalent to the VCSP with Boolean submodular valued constraints.

Remark 2.5.6. The class of valued constraints with submodular cost functions is the only non-trivial tractable class of optimisation problems in the dichotomy classification of Boolean VCSPs [CCJK06], and the only tractable class in the dichotomy classification of Max-CSPs for both 3-element domains [JKK06] and arbitrary finite domains allowing constant constraints [DJKK08] (and hence also for conservative Max-CSPs).

Remark 2.5.7. In all known dichotomy results for the Digraph Min-Cost Homomorphism problem, as listed in Example 2.2.28, the tractable cases admit either a *min-max* or a *k-min-max* ordering. A relation R admits a min-max ordering if, and only if, R is submodular. The concept of k-min-max ordering is a simple cyclic extension of submodularity which is well known to be tractable [GC08].

Cohen et al. have shown that VCSP instances with submodular constraints over an arbitrary finite domain can be reduced to the SFM [CCJK06], and hence can be solved in polynomial time. This tractability result has since been generalised to a wider class[6] of valued constraints over arbitrary finite domains known as tournament-pair valued constraints [CCJ08]

[6]The class of cost functions closed under a tournament-pair multimorphism is more general than the class of submodular cost functions if the range of the cost functions includes infinite costs [Coo08a].

An alternative approach to solving VCSP instances with submodular constraints, based on linear programming, can be found in [Coo08b].

2.6 Summary

We have introduced the VCSP framework and have given a long list of studied problems which can be easily described in this framework. We have also introduced the concept of expressibility and the related notion of algebraic properties of valued constraints. Chapter 3 deals with the expressive power of valued constraints in more depth. Finally, we have defined submodular functions and their basic properties.

Related work The area of constraint satisfaction has been very active in the last decade and there have been numerous extensions to the CSP framework. We refer the reader to standard textbooks [CKS01, Dec03, Apt03, RvBW06].

Let us mention at least two extensions which have been considered extensively in the literature.

The first extension deals with CSPs over infinite domains. Complexity classifications have been obtained for subsets of *Allen's interval algebra* [NB95, KJJ03], *equality constraint languages* [BK08a] and *temporal* CSPs [BK08b]. See also [BG08] and a nice recent survey by Bodirsky [Bod08].

The second extension deals with *quantified* CSPs. Creignou et al. have studied quantified CSPs over Boolean domains [CKS01]. More work on quantified CSPs was initiated by [BBJK03] and Chen's thesis [Che04], see also [Che08a] for an improved exposition of the collapsibility technique. A complexity classification has been obtained for *quantified equality constraint languages* [BC07] and *relatively quantified* CSPs [FK06, BC09] (that is, quantified CSPs with all unary constraints; in the case of non-quantified CSPs, these are known as conservative CSPs). See also [Che08b, Che09] for recent results in this area.

CHAPTER 3

Expressive Power of Valued Constraints

Never had any mathematical conversations with anybody,
because there was nobody else in my field.
Alonzo Church (1903–1995)

This chapter briefly discusses a recent result of Cohen, Cooper and Jeavons [CCJ06], and describes extensions of this result obtained by the author in collaboration with Dave Cohen, Martin Cooper and Pete Jeavons.

[CCJŽ09] D.A. Cohen, M.C. Cooper, P.G. Jeavons, and S. Živný. An Algebraic Characterisation of Complexity for Valued Constraint.
(Ongoing work.)
Earlier version of the first three authors in *Proceedings of the 12th International Conference on Principles and Practice of Contraint Programming (CP'06)*, volume 4204 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2006.

3.1 Introduction

It has been known for some time that the expressive power of crisp constraints is determined by certain algebraic operations called polymorphisms. Moreover, there is a Galois connection between the set of crisp constraints and the set of operations. This connection has been successfully used in the complexity analysis of crisp constraints [BKJ05].

In this chapter, we discuss the expressive power of valued constraints. We present a known characterisation of the expressive power of valued constraints by certain algebraic operations called fractional polymorphisms. We conjecture that more restrictive algebraic operations called multimorphisms determine the tractability of valued constraints, and also show that multimorphisms are not strong enough to characterise the

expressive power of valued constraints. We present results on the so-called *fractional clone theory*, and show connections to *linear programming*. We show a dual result to the results in [CCJ06]: determining whether a given fractional polymorphism belongs to a fractional clone is decidable.

This chapter is organised as follows. In Section 3.2, we present what is known about the expressive power of crisp constraints, and describe the concept of the indicator problem. In Section 3.3, we show that the question of whether a given cost function is expressible over a finite valued constraint language is decidable. In Section 3.4, we present some results from fractional clone theory and show some connections to linear programming. In Section 3.5, we discuss the expressibility of valued constraints versus the tractability of valued constraints. Finally, in Section 3.6, we conclude with related work and open questions.

3.2 Indicator problem

In this section, we discuss the well-known result that the expressive power of crisp constraints is characterised by certain algebraic operations called polymorphisms. We present the construction of an indicator problem, which is a universal construction for determining whether a given relation is expressible over a crisp constraint language, and also for determining all polymorphisms of a crisp constraint language. Finally, we show that there is a Galois connection between the set of relations and the set of operations.

Recall Theorem 2.4.6 which states that the expressive power of a crisp constraint language is fully characterised by its polymorphisms [Gei68, BKKR69, Jea98]. In other words, for a relation R and a crisp constraint language Γ, the following holds:

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

Remark 3.2.1. The "$\Rightarrow$" implication follows easily from the fact that expressibility preserves polymorphisms.

This result was obtained by showing that, for any crisp language (that is, set of relations), there is a *universal construction* which can be used to determine whether a relation is expressible in that language, as we now demonstrate.

Definition 3.2.2 (Indicator Problem)**.** Let Γ be a crisp constraint language over D. For any natural number n, we define the *indicator problem* for Γ of order n as the CSP instance $\mathcal{IP}(\Gamma, n)$ with set of variables D^n, each with domain D, and constraints $\{C_i\}_{1 \le i \le q}$, where $q = \sum_{R \in \Gamma} |R|^n$. For each $R \in \Gamma$, and for each sequence $t_1, t_2, \ldots, t_n$ of tuples from R, there is a constraint $C_i = \langle s_i, R \rangle$ with $s_i = \langle v_1, v_2, \ldots, v_m \rangle$, where m is the arity of R, and $v_j = \langle t_1[j], t_2[j], \ldots, t_n[j] \rangle$, $1 \le j \le m$.

Note that, for any crisp constraint language Γ over D, $\mathcal{IP}(\Gamma, n)$ has $|D|^n$ variables, and each corresponds to an n-tuple over D. A concrete example of an indicator problem is given below, and more examples can be found in [JCG96, JCG99].

Observation 3.2.3. It is not hard to see from Definition 3.2.2 and Definition 2.4.2 that the solutions to $\mathcal{IP}(\Gamma, n)$ are the polymorphisms of Γ of arity n [JCG97].

Combining Observation 3.2.3 with Theorem 2.4.6 gives:

Corollary 3.2.4. *Let Γ be a crisp constraint language over D. Furthermore, let $R = \{t_1, t_2, \ldots, t_n\}$ be a relation over D of arity m. Then R is expressible over Γ, $R \in \langle\Gamma\rangle$, if, and only if, R is equal to the solutions to $\mathcal{IP}(\Gamma, n)$ restricted to the variables $v_1, v_2, \ldots, v_m$, where $v_j = \langle t_1[j], t_2[j], \ldots, t_n[j]\rangle$, $1 \leq j \leq m$.*

Note that the choice of variables $v_1, v_2, \ldots, v_m$ might not be unique as different orderings of the tuples of R can result in different lists. We sketch the proof of Corollary 3.2.4 since it contains the idea behind the proof of Theorem 2.4.6.

Proof sketch. From Definition 3.2.2, if R is equal to the solutions to $\mathcal{IP}(\Gamma, n)$ restricted to some subset of variables, then R is expressible over Γ. On the other hand, assume that $R \in \langle\Gamma\rangle$, and denote by $\bar{R}$ the set of solutions to $\mathcal{IP}(\Gamma, n)$ restricted to the variables $v_1, v_2, \ldots, v_m$. It is enough to show that $R = \bar{R}$. By Observation 2.4.4, all projections are polymorphisms of all relations. Hence, by Observation 3.2.3, all projections of arity n are solutions of $\mathcal{IP}(\Gamma, n)$. Therefore, $R \subseteq \bar{R}$ from the choice of variables $v_1, v_2, \ldots, v_m$. If $R \neq \bar{R}$, then there must be a solution s to $\mathcal{IP}(\Gamma, n)$ whose restriction to $v_1, v_2, \ldots, v_m$ is not contained in R. By Observation 3.2.3, all solutions to $\mathcal{IP}(\Gamma, n)$ are polymorphisms of Γ, and so is s. But by Remark 3.2.1, the polymorphism s should be a polymorphism of R which is a contradiction provided $R \neq \bar{R}$. □

Remark 3.2.5. Richard Gault implemented a solver called POLYANNA[1] for the indicator problem [GJ04]. An interesting research problem is to investigate various symmetries in the indicator problem and try to make use of them in order to solve instances of the indicator problem more efficiently.

Example 3.2.6. Let $\Gamma = \{P, Q\}$ be a constraint language over $D = \{0, 1\}$, where $P = \{\langle 0, 1\rangle, \langle 1, 0\rangle\}$ and $Q = \{\langle 0\rangle\}$. Given the relation $R = \{\langle 0, 0\rangle, \langle 0, 1\rangle, \langle 1, 1\rangle\}$, the task is to determine whether R is expressible over Γ.

Since R consists of three tuples, we construct the indicator problem $\mathcal{IP}(\Gamma, 3)$ of order 3. The variables of $\mathcal{IP}(\Gamma, 3)$ are all 3-tuples over D. There are 8 3-tuples over D, and we denote them by v_0 to v_7 (see Figure 3.1). Since the relation P is binary, any two variables v_i and v_j, which represent the tuples $\langle v_{i1}, v_{i2}, v_{i3}\rangle$ and $\langle v_{j1}, v_{j2}, v_{j3}\rangle$, respectively, are constrained by P if, and only if, all three tuples $\langle v_{i1}, v_{j1}\rangle$, $\langle v_{i2}, v_{j2}\rangle$, and $\langle v_{i3}, v_{j3}\rangle$ belong to P. In our case the following pairs of variables are constrained by P: $\langle v_0, v_7\rangle$, $\langle v_1, v_6\rangle$, $\langle v_2, v_5\rangle$, and $\langle v_3, v_4\rangle$. The relation Q is unary and consists of just one tuple $\langle 0\rangle$. Therefore, only the variable v_0, which represents the tuple $\langle 0, 0, 0\rangle$, is constrained by Q. The construction is illustrated in Figure 3.1.

Now consider the tuples represented by the variables v_2 and v_3. These are $\langle 0, 1, 0\rangle$ and $\langle 0, 1, 1\rangle$, respectively. If you take these two tuples as columns of a matrix, then

[1] Available at: `http://www.comlab.ox.ac.uk/activities/constraints/software/index.html`

the rows of this matrix contain precisely the tuples from R, that is, $\langle 0,0\rangle$, $\langle 0,1\rangle$, and $\langle 1,1\rangle$. However, projecting the solutions to $\mathcal{IP}(\Gamma,3)$ onto variables v_2 and v_3 does not give R, as the tuple $\langle 1,0\rangle$ does not belong to R. (Some of the solutions to $\mathcal{IP}(\Gamma,3)$ are shown in Figure 3.1.) Hence R is not expressible over Γ. (Note that we could also obtain the same result by choosing, for instance, variables v_4 and v_6.)

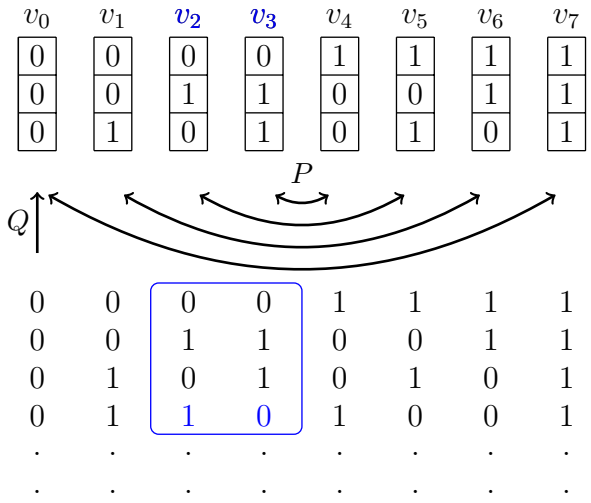

Figure 3.1: $\mathcal{IP}(\Gamma,3)$ (Example 3.2.6).

Recall from Notation 2.4.5 that, for a crisp constraint language Γ, we denote by $\mathsf{Pol}(\Gamma)$ the set of all polymorphisms of Γ, that is,

$$\mathsf{Pol}(\Gamma) = \{f \mid \forall R \in \Gamma, f \text{ is a polymorphism of } R\}.$$

In this section, we shall use the word operation for any mapping from D^k to D.

Notation 3.2.7. For a set of operations O, we use $\mathsf{Inv}(O)$ to denote the set of relations having all operations in O as a polymorphism, that is,

$$\mathsf{Inv}(\Gamma) = \{R \mid \forall f \in O, f \text{ is a polymorphism of } R\}.$$

Observation 3.2.8. The result of Theorem 2.4.6 can be equivalently stated as follows: for any crisp constraint language Γ, it holds that $\langle\Gamma\rangle = \mathsf{Inv}(\mathsf{Pol}(\Gamma))$.

Definition 3.2.9 (Galois connection [DW02])**.** A *Galois connection* between two sets A and B is a pair $\langle F, G\rangle$ of mappings between the power sets $\mathcal{P}(A)$ and $\mathcal{P}(B)$, $F : \mathcal{P}(A) \to \mathcal{P}(B)$ and $G : \mathcal{P}(B) \to \mathcal{P}(B)$, such that for all $X, X' \subseteq A$ and all $Y, Y' \subseteq B$ the following conditions are satisfied: $X \subseteq X' \Rightarrow F(X) \supseteq F(X')$, and $Y \subseteq Y' \Rightarrow G(Y) \supseteq G(Y')$; $X \subseteq G(F(X))$, and $Y \subseteq F(G(Y))$.

Notation 3.2.10. For any finite domain D, we denote by $\mathbf{R}_D$ the set of all relations over D, and we denote by $\mathbf{O}_D$ the set of all operations over D.

The following easy result shows that the $\mathsf{Pol}(\cdot)$ and $\mathsf{Inv}(\cdot)$ operators give rise to an instance of a *Galois connection* between $\mathbf{R}_D$ and $\mathbf{O}_D$ for any finite domain D.

Proposition 3.2.11 ([PK79])**.** *If Γ is a set of relations over D and O is a set of operations over D, then*

1. $O_1 \subseteq O_2 \subseteq \mathbf{O}_D \;\Rightarrow\; \mathsf{Inv}(O_1) \supseteq \mathsf{Inv}(O_2)$.
2. $\Gamma_1 \subseteq \Gamma_2 \subseteq \mathbf{R}_D \;\Rightarrow\; \mathsf{Pol}(\Gamma_1) \supseteq \mathsf{Pol}(\Gamma_2)$.
3. $\Gamma \subseteq \mathsf{Inv}(\mathsf{Pol}(\Gamma))$.
4. $O \subseteq \mathsf{Pol}(\mathsf{Inv}(O))$.

Recall that for a crisp constraint language $\Gamma \subseteq \mathbf{R}_D$, we denote by $\langle\Gamma\rangle$ the set of relations which are expressible over Γ. The set $\langle\Gamma\rangle$ is also known as a *relational clone* [PK79]. For a set of operations $O \subseteq \mathbf{O}_D$, we denote by $\langle O\rangle$ the set of operations from O closed under composition and containing all projections. The set $\langle O\rangle$ is known as a *clone of operations*, or just a *clone* [PK79].

Corollary 3.2.12 (of Theorem 2.4.6)**.** *For any two crisp constraint languages Γ_1 and Γ_2, and any two sets of operations O_1 and O_2,*

1. $\langle\Gamma_1\rangle = \langle\Gamma_2\rangle \;\Leftrightarrow\; \mathsf{Pol}(\Gamma_1) = \mathsf{Pol}(\Gamma_2)$.
2. $\langle O_1\rangle = \langle O_2\rangle \;\Leftrightarrow\; \mathsf{Inv}(O_1) = \mathsf{Inv}(O_2)$.

The situation is summarised in Figure 3.2.

Remark 3.2.13. Post completely described the lattice of relational clones and clones over a two-element domain [Pos41]. This description has been heavily used recently to obtain dichotomy complexity classifications for various problems in computer science and artificial intelligence that can be modelled over Boolean domains. For more details, see [BCRV03, BCRV04].

More on clone theory can be found in [PK79, DW02].

3.3 Weighted indicator problem

In this section, we show that for valued constraints there is also a universal construction to determine whether a given cost function is expressible over a valued constraint language. We briefly describe the result that the expressive power of valued constraints is determined by certain algebraic operations called fractional polymorphisms.

Consider the following problem: given a cost function ϕ of arity m over a domain D, is ϕ expressible over a valued constraint language Γ? We show that this problem is decidable for every finite Γ. First we prove an upper bound on the number of extra variables needed to express ϕ over Γ.

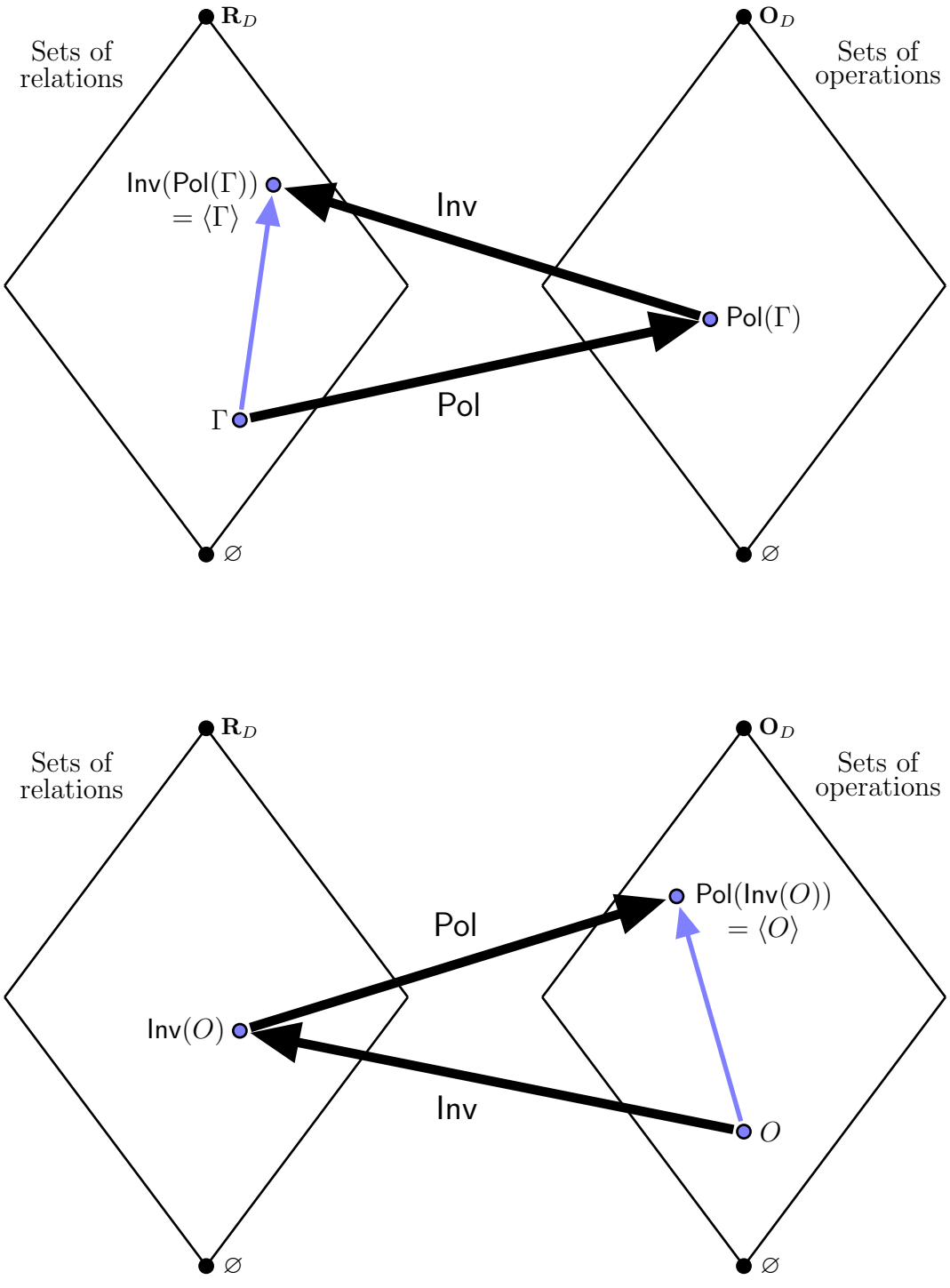

Figure 3.2: Galois connection between $\mathbf{R}_D$ and $\mathbf{O}_D$.

Proposition 3.3.1 ([CCJ06]). *If a cost function $\phi : D^m \to \overline{\mathbb{R}}$ is expressible over Γ, then ϕ is expressible over Γ using at most $|D|^{|D|^m}$ hidden variables.*

Proof. If $\phi \in \langle\Gamma\rangle$, then by Definition 2.3.1, there is a gadget $\langle\mathcal{P}, \mathbf{v}\rangle$, where $\mathbf{v} = \langle v_1, \ldots, v_m\rangle$, for expressing ϕ over Γ. For the gadget $\langle\mathcal{P}, \mathbf{v}\rangle$ to express ϕ, it has to define ϕ on each of the $|D|^m$ different assignments to $\mathbf{v}$. Let each of these $|D|^m$ assignments be extended to a complete assignment to all variables of $\mathcal{P}$ (including hidden variables) in a way that minimises the total cost. For each hidden variable v of $\langle\mathcal{P}, \mathbf{v}\rangle$, we can use the list of $|D|^m$ values assigned to v by these complete assignments to label the variable v. If there are more than $|D|^{|D|^m}$ hidden variables, then two of them will receive the same label. However, this implies that one of the two is redundant, as all constraints involving that variable can replace it with the other variable without changing the overall cost. Hence we require at most $|D|^{|D|^m}$ distinct hidden variables to express ϕ. □

From this bound on the number of extra variables in a gadget for ϕ over Γ we obtain a decidability result. The idea is to try all possible constraints on all possible subsets of variables, and use linear programming to determine whether there is a combination of these constraints which works.

Theorem 3.3.2 ([CCJ06]). *For a given finite valued constraint language Γ, and a cost function ϕ defined over D, the question of whether ϕ is expressible over Γ is decidable.*

Proof sketch. In order to simplify the presentation, we assume that ϕ is a finite-valued cost function. We show how to determine whether there is a gadget for ϕ over Γ; that is, whether there is a $\mathsf{VCSP}(\Gamma)$ instance $\mathcal{P} = \langle V, D, \mathcal{C}\rangle$ and a tuple of variables $\mathbf{v}$ such that $\phi = \pi_{\mathbf{v}}(\mathcal{P})$. By Proposition 3.3.1, $\mathcal{P}$ has at most $K = |D|^{|D|^m}$ extra variables, where m is the arity of ϕ. Let V be the set of K variables, each associated with a different $|D|^m$-tuple of values from D. Let E be the $|D|^m \times K$ matrix whose columns are all possible $|D|^m$ tuples of values from D (or equivalently, variables from V). Observe that there is a $|D|^m \times m$ submatrix E' of E consisting of m columns of E such that the rows of E' correspond to all possible m-tuples of values from D. We let $\mathbf{v}$ be the list of variables corresponding to the columns of E'.

Let $\mathcal{A}$ be the set of all assignments of values from D to the variables from V. Clearly, $|\mathcal{A}| = |D|^K$. We choose $|D|^m$ assignments from $\mathcal{A}$ that correspond to the rows of the matrix E and denote them $\mathcal{A}'$.

Let $\rho \in \Gamma$ be a cost function of arity k. For an assignment $s \in \mathcal{A}$ and a list of variables $\mathbf{u}$ of length k, recall from Definition 2.1.7 that we denote by $\rho(s(\mathbf{u}))$ the value of ρ on the list of variables $\mathbf{u}$ assigned by s.

The idea is that if ϕ is expressible over Γ, then there is a set of constraints $\mathcal{C}$ such that $\phi = \pi_{\mathbf{v}}(\mathcal{P})$, where $\mathcal{P} = \langle V, D, \mathcal{C}\rangle$. It remains to show what the set of constraints $\mathcal{C}$ is. And this is where linear programming plays its crucial role.

Let $\mathcal{C}$ be the list of all possible constraints from Γ applied on variables from V. In other words, $\mathcal{C} = \langle C_1, \ldots, C_q\rangle$ is an arbitrary but fixed order of the following finite set:

$$\{\langle\mathbf{u}, \rho\rangle \mid \rho \in \Gamma \text{ of arity } k, \text{ and } \mathbf{u} \text{ is a list of } k \text{ variables from } V\}.$$

We denote by $\mathbf{u}_i = \mathbf{u}$ and $\rho_i = \rho$, where $C_i = \langle \mathbf{u}, \rho \rangle$. Clearly,

$$q = \sum_{\rho \in \Gamma \text{ of arity } k} K^k$$

We define a system of linear equations and inequalities as follows.
For each $s \in \mathcal{A} \setminus \mathcal{A}'$,

$$\sum_{i=1}^{q} x_i \rho_i(s(\mathbf{u}_i)) \;\geq\; \phi(s(\mathbf{v})) + x_0.$$

For each $s \in \mathcal{A}'$,

$$\sum_{i=1}^{q} x_i \rho_i(s(\mathbf{u}_i)) \;=\; \phi(s(\mathbf{v})) + x_0.$$

Note that the variable x_i represents whether the constraint C_i is used in the gadget $\mathcal{P}$: if $x_i = 0$, then the constraint $\mathcal{C}_i$ is not used; if $x_i > 0$, then x_i gives the multiplicity of the constraint C_i. The variable x_0 represents an additive constant up to which the gadget expresses ϕ.

From the construction of the system, ϕ is expressible over Γ if, and only if, there is a non-negative solution to this system. But this question is decidable [Sch86], see also [AK04]. □

Remark 3.3.3. Theorem 3.3.2 can be extended from finite-valued cost functions to general cost functions [CCJ06]. The construction sketched above is known as the *weighted indicator problem*.

Example 3.3.4. Let $\Gamma = \{\mu, \psi\}$ be the valued constraint language consisting of two cost functions defined over the Boolean domain $D = \{0, 1\}$ as follows:

$$\mu(x) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = 0, \\ 1 & \text{if } x = 1, \end{cases}$$

and

$$\psi(x, y) \stackrel{\text{def}}{=} \begin{cases} -1 & \text{if } x = y = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Let ϕ be the ternary cost function defined as follows:

$$\phi(x, y, z) \stackrel{\text{def}}{=} \begin{cases} -1 & \text{if } x = y = z = 1, \\ 0 & \text{otherwise.} \end{cases}$$

The question is whether ϕ is expressible over Γ, that is, whether $\phi \in \langle \Gamma \rangle$. In order to answer this question, we build an instance of the weighted indicator problem as described in the sketched proof of Theorem 3.3.2. The arity m of ϕ is 3, and hence if ϕ is expressible over Γ, then ϕ is expressible with at most $K = |D|^{|D|^m} = 2^{2^3} = 2^8 = 256$

variables, by Proposition 3.3.1. Each variable is uniquely identified by an 8-tuple of values from $\{0,1\}$. We denote by V the set of all such variables with the domain $\{0,1\}$.

We denote by $\mathbf{v} = \langle v_1, v_2, v_3 \rangle$ the list of 3 variables, whose corresponding 8-tuples are $t_1 = \langle 0,0,0,0,1,1,1,1 \rangle$, $t_2 = \langle 0,0,1,1,0,0,1,1 \rangle$, and $t_3 = \langle 0,1,0,1,0,1,0,1 \rangle$ respectively. Consider the matrix whose columns are tuples t_1, t_2, and t_3. The rows of this matrix are all possible 3-tuples over $\{0,1\}$. The intuition is that we try to find a gadget $\mathcal{P}$ for ϕ over Γ which expresses ϕ on the variables v_1, v_2, and v_3, that is, $\phi = \pi_{\langle v_1, v_2, v_3 \rangle}(\mathcal{P})$.

Let E be an 8×256 matrix whose columns are the tuples corresponding to variables from V in some fixed order.

We denote by $\mathcal{A}'$ the set of 8 assignments of variables from V which are defined by the rows of the matrix E. The intuition is that for every possible assignment s of the variables v_1, v_2, and v_3, we are looking for an assignment s' in $\mathcal{A}'$ which agrees with s (on v_1, v_2 and v_3), and the cost of s' is equal to $\phi(v_1, v_2, v_3)$ (up to an additive constant). We denote by $\mathcal{A}$ all assignments of variables from V. Clearly, $|\mathcal{A}| = 2^{256}$.

Now we want to add all possible constraints involving cost functions from Γ. The unary cost function μ can be applied to any of the 256 variables. The binary cost function ψ can be applied to any pair of (not necessarily distinct) variables. Since ψ is symmetric, this gives $\binom{256}{2} + 256$ constraints. In total, we get $2 * 256 + \binom{256}{2} = 33152$ constraints. Hence we have 33152 variables x_i which represent whether or not the i-th constraint is used ($x_i > 0$) or not ($x_i = 0$); in the former case, the value of x_i represents the multiplicity of the i-th constraint. We then can build a system of linear equations and inequalities as described in the sketch of the proof of Theorem 3.3.2.

In this particular case, it is not difficult to find a solution to the system of linear equations and inequalities described above. Let y be the variable corresponding to the 8-tuple $\langle 0,0,0,0,0,0,0,1 \rangle$. We claim that assigning the value 2 to the constraint $\langle \langle y \rangle, \mu \rangle$ (represented by a variable in our system), assigning the value 1 to the constraints $\langle \langle y, x_1 \rangle, \psi \rangle$, $\langle \langle y, x_2 \rangle, \psi \rangle$, and $\langle \langle y, x_3 \rangle, \psi \rangle$, and finally assigning the value 0 to the additive constant $x_0 = 0$ and all other variables is a solution to our system. For any assignment of the variables v_1, v_2 and v_3, setting y to 0 results in total cost 0. If all v_1, v_2 and v_3 are assigned 1, setting y to 1 results in total cost -1. For any other assignment of v_1, v_2 and v_3, setting y to 1 results in total cost ≥ 0. This corresponds exactly to the definition of the cost function ϕ. This solution gives a gadget for expressing ϕ over Γ using only one extra variable.

Recall Theorem 2.4.19, which states that the expressive power of a valued constraint language satisfying certain conditions is fully characterised by its feasibility polymorphisms and fractional polymorphisms [CCJ06]. In other words, for a cost function ϕ and a valued constraint language Γ, such that Γ contains a constant function and is closed under scaling and the feasibility operator, the following holds:

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

Remark 3.3.5. The "⇒" implication follows easily from the fact that expressibility preserves feasibility polymorphisms and fractional polymorphisms [CCJ06], see also [CCJK06].

Remark 3.3.6. We remark on the assumptions of Theorem 2.4.19. Notice that it is not a restrictive assumption that every valued constraint language Γ contains a constant function and is closed under scaling. In practice, this corresponds to adding a finite constant which does not alter the relative costs, and to taking more copies of the same constraint. Therefore, this does not change the complexity of solving VCSP instances over Γ.

We now discuss why it is necessary to assume that Γ is closed under the feasibility operator in order to prove equivalence in Theorem 2.4.19. Recall from Notation 2.1.6 that, for a valued constraint language Γ, $\mathsf{Feas}(\Gamma)$ is the set of cost functions where finite costs are replaced by zero. If $\mathcal{F}$ is a fractional polymorphism of Γ, then $\mathcal{F}$ is also a fractional polymorphism of $\mathsf{Feas}(\Gamma)$. And clearly, any feasibility polymorphism of $\mathsf{Feas}(\Gamma)$ is a feasibility polymorphism of Γ. Hence for any valued constraint language Γ, $\mathsf{FPol}(\Gamma) \subseteq \mathsf{FPol}(\mathsf{Feas}(\Gamma))$ and $\mathsf{fPol}(\Gamma) \subseteq \mathsf{fPol}(\mathsf{Feas}(\Gamma))$. But this is true for any Γ independently of whether or not $\mathsf{Feas}(\Gamma) \in \langle\Gamma\rangle$, so every valued constraint language Γ satisfies the right hand side of the equivalence in Theorem 2.4.19 for $\mathsf{Feas}(\Gamma)$ (that is, $\mathsf{FPol}(\Gamma) \subseteq \mathsf{FPol}(\mathsf{Feas}(\Gamma))$ and $\mathsf{fPol}(\Gamma) \subseteq \mathsf{fPol}(\mathsf{Feas}(\Gamma))$), but not every valued constraint language Γ satisfies $\mathsf{Feas}(\Gamma) \in \langle\Gamma\rangle$.

Fractional polymorphisms on their own characterise the expressive power of finite-valued cost functions and, as shown in Theorem 2.4.6, feasibility polymorphisms on their own characterise the expressive power of crisp cost functions which take only zero and infinite costs.

The proof of Theorem 2.4.19 given in [CCJ06] is based on an application of Farkas' Lemma [Sch86] and uses the concept of the weighted indicator problem. For a given ϕ and Γ, as sketched above, a system of linear equations and inequalities is built such that this system has a solution if, and only if, ϕ is expressible over Γ. If this system does not have a solution, then Farkas' Lemma guarantees a certificate of unsolvability. Cohen et al. have shown that the certificate of unsolvability can be turned into a fractional polymorphism $\mathcal{F}$ such that $\mathcal{F} \in \mathsf{fPol}(\Gamma)$, but $\mathcal{F} \notin \mathsf{fPol}(\{\phi\})$ [CCJ06].

3.4 Fractional clone theory

In this section, we discuss the so-called fractional clone theory. The goal is to provide a theory of algebraic operations which satisfy certain closure properties and which are related to the set of cost functions via a Galois connection. Similar theory has played a crucial role in the complexity analysis of crisp constraints, and we believe that such a theory will play a similar role in the complexity analysis of valued constraints.

In this section, we will deal primarily with finite-valued cost functions so that we can restrict our attention to fractional polymorphisms as operations. We will mention an extension to general cost functions at the end of this section.

3.4.1 Fractional polymorphisms

Recall from Notation 2.4.15 that, for a valued constraint language Γ, we denote by $\mathsf{fPol}(\Gamma)$ the set of all fractional polymorphisms of Γ, that is,

$$\mathsf{fPol}(\Gamma) = \{f \mid \forall \phi \in \Gamma, f \text{ is a fractional polymorphism of } \phi\}.$$

Notation 3.4.1. For a set of weighted mappings O, we use $\mathsf{Imp}(O)$ to denote the set of cost functions that have weighted mappings in O as fractional polymorphisms, that is,

$$\mathsf{Imp}(O) = \{\phi \mid \forall \mathcal{F} \in O, \mathcal{F} \text{ is a fractional polymorphism of } \phi\}.$$

The notation $\mathsf{Imp}(O)$ is an abbreviation for "improved by O": the cost functions for which weighted mappings from O are fractional polymorphisms are precisely those cost functions whose aggregated value is "improved" (that is, lowered, or left unchanged) by weighted mappings from O, as described in Figure 2.4.

Notation 3.4.2. For any finite domain D, we denote by $\mathbf{F}_D$ the set of all finite-valued cost functions over D, and we denote by $\mathbf{O}_D^f$ the set of all weighted mappings over D.

In this section, we shall use the word operation for any weighted mapping (as defined in Definition 2.4.9).

The following easy result follows immediately from the definitions, and shows that the $\mathsf{fPol}(\cdot)$ and $\mathsf{Imp}(\cdot)$ operators give rise to an instance of a *Galois connection* between $\mathbf{F}_D$ and $\mathbf{O}_D^f$ for any finite domain D.

Proposition 3.4.3. *If Γ is a set of cost functions over D and O is a set of operations over D, then*

1. $O_1 \subseteq O_2 \subseteq \mathbf{O}_D^f \;\Rightarrow\; \mathsf{Imp}(O_1) \supseteq \mathsf{Imp}(O_2)$.
2. $\Gamma_1 \subseteq \Gamma_2 \subseteq \mathbf{F}_D \;\Rightarrow\; \mathsf{fPol}(\Gamma_1) \supseteq \mathsf{fPol}(\Gamma_2)$.
3. $\Gamma \subseteq \mathsf{Imp}(\mathsf{fPol}(\Gamma))$.
4. $O \subseteq \mathsf{fPol}(\mathsf{Imp}(O))$.

Recall from Observation 2.4.18 that a multi-projection is a mapping from D^k to D^k that only permutes the set of its arguments.

Observation 3.4.4. Let D be an arbitrary finite domain, and let $\Gamma \subseteq \mathbf{F}_D$ be an arbitrary set of finite-valued cost functions over D. Then $\mathsf{fPol}(\Gamma)$ contains all multi-projections. In other words, the set of operations which are fractional polymorphisms of all finite-valued cost functions is precisely the set of all multi-projections.

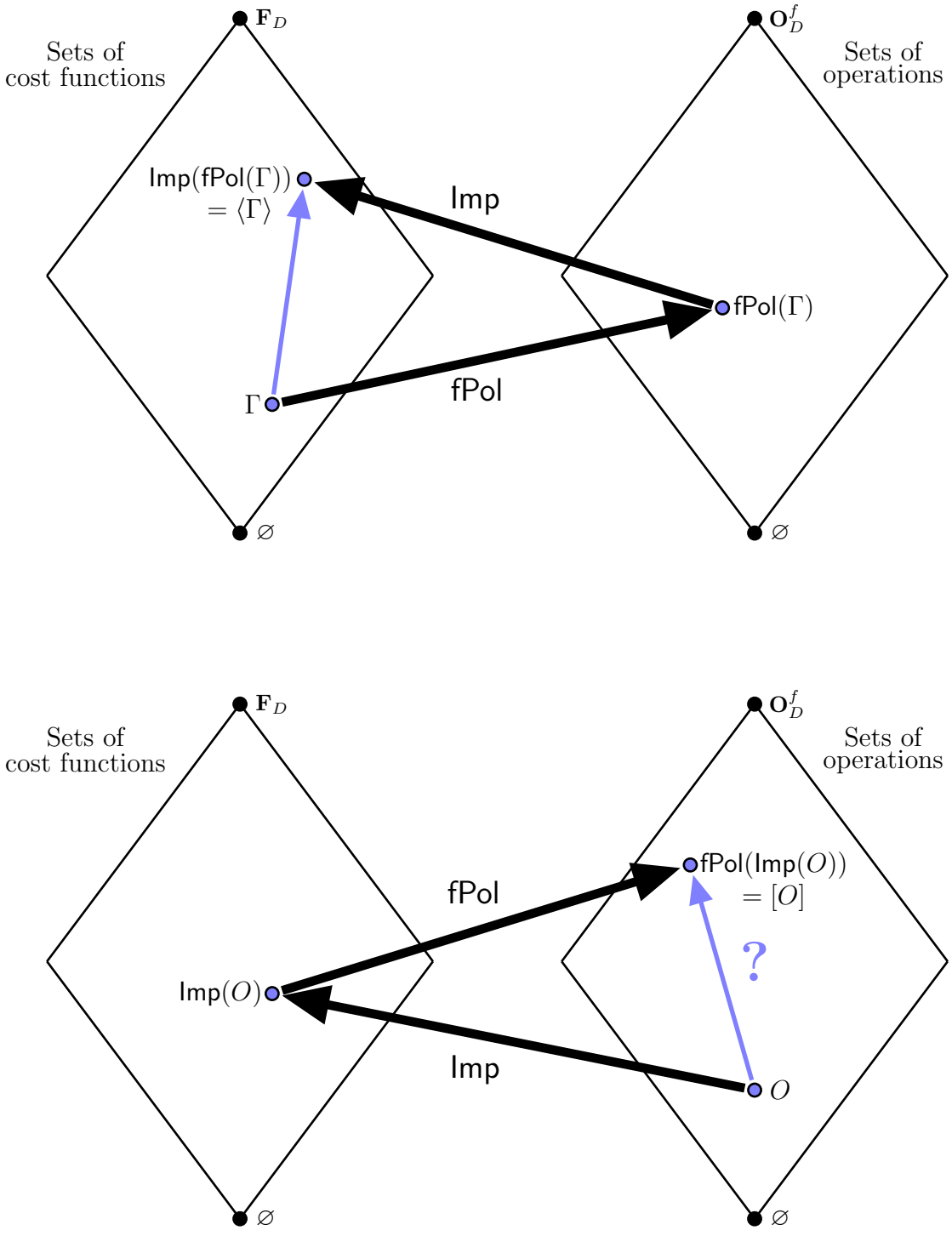

Figure 3.3: Galois connection between $\mathbf{F}_D$ and $\mathbf{O}_D^f$.

In the set of cost functions, we have a closure operator which corresponds to expressibility. This closure operator with the Galois connection between $\mathbf{F}_D$ and $\mathbf{O}_D^f$ shows, for instance, that for a given valued constraint language Γ the fewer fractional polymorphisms Γ has, the harder Γ is. In fact, Theorem 2.4.19 can be restated equivalently as follows:

$$\langle\Gamma\rangle = \mathsf{Imp}(\mathsf{fPol}(\Gamma)).$$

However, we do not know what the natural closure operator is in the set of operations. (This is described by the question mark on the right side in Figure 3.3.)

Notation 3.4.5. For a set O of operations over a fixed finite domain D, we denote by $[O]$ the *fractional clone* generated by O and define $[O] = \mathsf{fPol}(\mathsf{Imp}(O))$.

The situation is summarised in Figure 3.3. We do not know how to get from O to $[O]$. However, using the Galois connection we show now that the question of whether a given operation belongs to a particular fractional clone is decidable.

Theorem 3.4.6. *For a given operation O and a fractional clone $[\Omega]$ generated by a finite set of operations $\Omega = \{O_1, \ldots, O_q\}$, the question of whether O belongs to $[\Omega]$ is decidable.*

Proof. Let k be the arity of O; that is, O is a weighted mapping $\{\langle r_1, f_1\rangle, \ldots, \langle r_n, f_n\rangle\}$ such that each f_i is a distinct function from D^k to D, each r_i is a positive rational number and $\sum_{i=1}^n r_i = k$ (see Definition 2.4.10).

Now $O \notin [\Omega]$ if, and only if, there is a finite-valued cost function ϕ such that $O_i \in \mathsf{fPol}(\{\phi\})$ for every $1 \leq i \leq q$, but $O \notin \mathsf{fPol}(\{\phi\})$.

First we show that if there is such a ϕ (we call it a separating cost function), then there is one of arity at most $m = |D|^k$. Assume that ϕ is of arity strictly bigger than m. As there are exactly m different k-tuples over D, any tableau (recall Figure 2.4) showing that $O_i \in \mathsf{fPol}(\{\phi\})$, $1 \leq i \leq q$, and that $O \notin \mathsf{fPol}(\{\phi\})$ has at least one column which occurs twice. However, this column can be removed and the arity of ϕ decreased by 1 by identifying the two arguments corresponding to the two columns. Clearly, if there is a separating cost function ϕ of arity strictly smaller than m, then there is a separating cost function of arity exactly m: we just add dummy variables. Hence we can assume that ϕ is of arity exactly m.

Now we can turn the question of the existence of a separating cost function into a system of linear inequalities. We are looking for $|D|^m$ values (costs of ϕ on all possible assignments) such that for all $1 \leq i \leq q$, $O_i \in \mathsf{fPol}(\{\phi\})$, and $O \notin \mathsf{fPol}(\{\phi)\}$. But this is easy as showing that $O_i \in \mathsf{fPol}(\{\phi\})$ is just a question of satisfying a system of linear inequalities for all possible tableaux, by Definition 2.4.10. Similarly, $O \notin \mathsf{fPol}(\{\phi\})$ can be expressed as one linear inequality corresponding to the tableau with m different k-tuples over D and the inequality sign the opposite way to Definition 2.4.10. This finishes the proof as the question of whether there is a solution to a system of linear inequalities is decidable [Sch86]. □

The following example illustrates the technique described in the proof of Theorem 3.4.6.

Example 3.4.7. Let $O_1 = \{\langle 1, \text{MIN}\rangle, \langle 1, \text{MAX}\rangle\}$, $O = \{\langle 2, \text{MAX}\rangle\}$, and $D = \{0, 1\}$. In order to determine, whether $O \in [O_1]$, we build a system of linear inequalities as in the proof of Theorem 3.4.6. We look for a separating cost function ϕ of arity $m = |D|^2 = 4$. Hence we have $|D|^4 = 16$ variables $x_{0000}, x_{0001}, \ldots, x_{1111}$ corresponding to the values of ϕ on 16 different assignments. There are $16 * 16 = 256$ inequalities that make sure that $\{\langle 1, \text{MIN}\rangle, \langle 1, \text{MAX}\rangle\} \in \mathsf{fPol}(\{\phi\})$:

$$x_{ijkl} + x_{mnop} \;\geq\; x_{abcd} + x_{uvyz},$$

where $a = \min(i, m)$, $b = \min(j, n)$, $c = \min(k, o)$, $d = \min(l, p)$, and $u = \max(i, m)$, $v = \max(j, n)$, $y = \max(k, o)$, $z = \max(l, p)$.

Another inequality makes sure that $\{\langle 2, \text{MAX}\rangle\} \notin \mathsf{fPol}(\{\phi\})$. According to the proof of Theorem 3.4.6, the tableau consists of 4 2-tuples over $\{0, 1\}$. Hence, the required inequality is

$$x_{0011} + x_{0101} \;<\; x_{0111} + x_{0111},$$

where $T = \binom{0011}{0101}$ on the left hand side corresponds to 4 different 2-tuples (column-wise), and $\binom{0111}{0111}$ on the right hand side is the application of O on T.

One solution to this system is $x_{00..} = 0$, and $x_{01..} = x_{10..} = x_{11..} = 1$. Notice that ϕ is affectively binary as it only depends on its first two arguments: $\phi(x, y, ., .) = 0$ if $x = y = 0$, and 1 otherwise. It is straightforward to check that this is indeed a solution to the system; that is, $\{\langle 1, \text{MIN}\rangle, \langle 1, \text{MAX}\rangle\} \in \mathsf{fPol}(\{\phi\})$, but $\{\langle 2, \text{MAX}\rangle\} \notin \mathsf{fPol}(\{\phi\})$.

Given a finite set of operations Ω and an operation O, if $O \notin [\Omega]$, Theorem 3.4.6 tells us more than that: the system of inequalities does not have a solution, and hence by Farkas' Lemma [Sch86], there is a certificate on unsolvability. At the moment, it is not clear how to turn this certificate into a closure operator in the set of operations.

3.4.2 Multimorphisms

We now turn to multimorphisms, which form a subclass of fractional polymorphisms. In this section, we shall use the word operation for any mapping from D^k to D^k.

Notation 3.4.8. For any finite domain D, we denote by $\mathbf{O}_D^m$ the set of all mappings from D^k to D^k for some k.

Remark 3.4.9. Similarly to Proposition 3.4.3, it can be easily shown that there is a Galois connection between $\mathbf{F}_D$ and $\mathbf{O}_D^m$ over any finite D given by the $\mathsf{Mul}(\cdot)$ operator (from Notation 2.4.15) and the $\mathsf{Imp}(\cdot)$ operator. Although we still have the expressibility closure operator for cost functions, because multimorphisms on their own are not known[2] to characterise the expressive power of valued constraints, we do not know what the relationship is between $\Gamma \subseteq \mathbf{F}_D$ and $\mathsf{Imp}(\mathsf{Mul}(\Gamma))$. Similarly, we do not know what the relationship is between $O \subseteq \mathbf{O}_D^m$ and $\mathsf{Mul}(\mathsf{Imp}(O))$. (This is illustrated by two question marks in Figure 3.4.)

[2]In fact, we will see in Section 3.5 that multimorphism *do not* characterise the expressive power of valued constraints.

Notation 3.4.10. For a set O of mappings from D^k to D^k over a fixed finite domain D, we define the *multi-clone* generated by O as $[O]_m = \mathsf{Mul}(\mathsf{Imp}(O))$.

Similarly to Observation 3.4.4, we obtain

Observation 3.4.11. For any finite domain D, the set of all multimorphisms of all finite-valued cost functions over D, $\mathsf{Mul}(\mathbf{F}_D)$, is precisely the set of all multi-projections.

We now observe some basic properties of multi-clones.

Definition 3.4.12. We say that a class O of mappings is *closed under extension* if for every $F = \langle f_1, \ldots, f_k \rangle : D^k \to D^k$ in O, the function $F' : D^{k+1} \to D^{k+1}$ is also in O, where $F' = \langle f_1, \ldots, f_k, \pi_{k+1} \rangle$.

Definition 3.4.13. We say that a class O of mappings is *closed under permutation* if whenever $\mathcal{G} = \langle g_1, \ldots, g_k \rangle : D^k \to D^k$ is in O and $\pi, \sigma : \{1, \ldots, k\} \to \{1, \ldots, k\}$ are permutations, then $\mathcal{F} = \langle f_1, \ldots, f_k \rangle : D^k \to D^k$ is also in O, where $f_i(x_1, \ldots, x_k) = g_{\pi(i)}(x_{\sigma(1)}, \ldots, x_{\sigma(k)})$, $1 \leq i \leq k$.

Definition 3.4.14. We say that a class O of mappings is *closed under (function) composition* if for every $G, F : D^k \to D^k$ in O, the composition of G and F is also in O.

Observation 3.4.15. If follows from Definition 2.4.11, that for any valued constraint language Γ, the set $\mathsf{Mul}(\Gamma)$ is closed under extension, permutation and composition.

Remark 3.4.16. This section has dealt with finite-valued cost functions only. We have seen in Theorem 2.4.19 that the expressive power of valued constraints is characterised by fractional polymorphisms and feasibility polymorphisms; feasibility polymorphisms characterise the expressive power of crisp constraints. For the crisp case, we have a Galois connection with closure operators for both relations and feasibility polymorphisms (Section 3.2). Hence, if we could find a closure operator for fractional polymorphisms, which characterises the expressive power of finite-valued cost functions, it would be easy to join it with feasibility polymorphisms to get a theory for general cost functions: operations would be pairs where the first component is a set of fractional polymorphisms, and the second component is a set of feasibility polymorphisms.

3.5 Expressibility versus tractability

We have seen in Theorem 2.4.19 that fractional polymorphisms and feasibility polymorphisms together characterise the expressive power of valued constraints. Therefore, in order to study the tractability of valued constraint languages we only need to understand the fractional polymorphisms and feasibility polymorphisms of valued constraints.

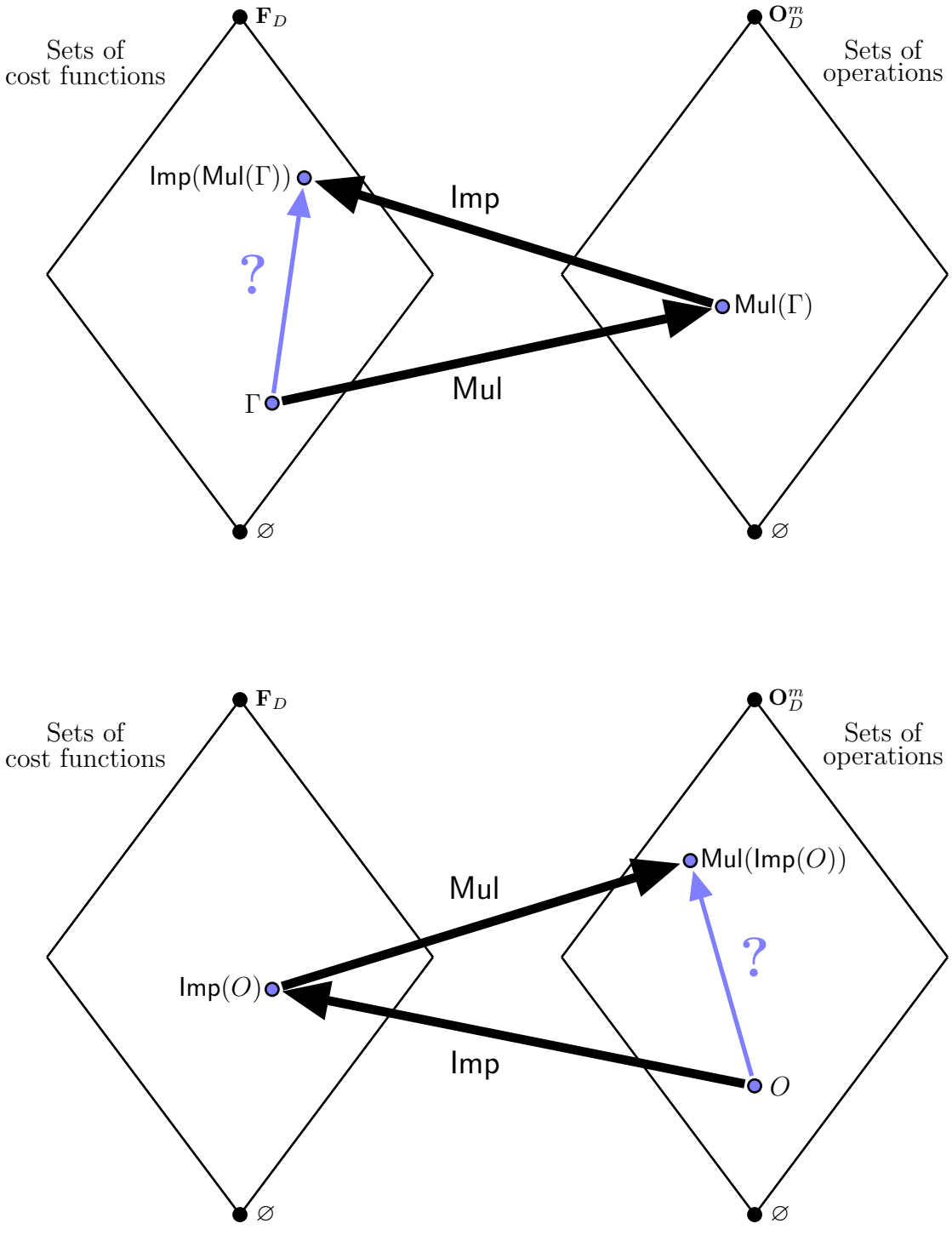

Figure 3.4: Galois connection between $\mathbf{F}_D$ and $\mathbf{O}_D^m$.

As we show in this section, Theorem 2.4.19 implies that non-trivial fractional polymorphisms are necessary for tractability of valued constraints. We conjecture that in fact a stronger result holds: non-trivial multimorphisms are necessary for tractability of valued constraints. On the other hand, we show that multimorphisms on their own are not strong enough to characterise the expressive power of valued constraints. We also show that non-trivial fractional polymorphisms are not sufficient for tractability.

Theorem 3.5.1 ([CCJ06])**.** *If all fractional polymorphisms of a valued constraint language* Γ *are multi-projections,*[3] *then* Γ *is intractable.*

Proof. Suppose that every fractional polymorphism of Γ is a multi-projection. By Observation 2.4.18, we have that $\mathsf{fPol}(\Gamma) \subseteq \mathsf{fPol}(\{\phi_{\mathsf{xor}}\})$, where ϕ_{xor} is from Example 2.2.10. Since $\mathsf{Feas}(\phi_{\mathsf{xor}})$ is the cost function whose costs are all zero, it follows that ϕ_{xor} has all possible feasibility polymorphisms. Hence $\mathsf{FPol}(\Gamma) \subseteq \mathsf{FPol}(\{\phi_{\mathsf{xor}}\})$. By Theorem 2.4.19, $\phi_{\mathsf{xor}} \in \langle\Gamma\rangle$. Therefore, by Example 2.2.10 and Theorem 2.4.21, Γ is intractable. □

All known tractable valued constraint languages can be characterised by the special kinds of fractional polymorphisms known as multimorphisms. This raises the question of whether multimorphisms alone are sufficient to characterise the expressive power of valued constraints. Unfortunately, the answer is negative.

Theorem 3.5.2 (Dave Cohen)**.** *For every finite domain* D*, there is a cost function* ϕ_D *such that* ϕ_D *has a non-trivial unary fractional polymorphism, but the only multimorphisms of* ϕ_D *are multi-projections.*

Theorem 3.5.2 shows that multimorphisms are not strong enough to characterise the expressive power of valued constraints. More concretely, ϕ_D cannot express all cost functions because, by Remark 3.3.5, expressibility preserves fractional polymorphisms, and by Observation 3.4.4, the only fractional polymorphisms of all cost functions are multi-projections. Hence there is a cost function ψ such that $\psi \notin \langle\{\phi_D\}\rangle$, but there is no separating multimorphism $\mathcal{F}$ such that $\mathcal{F} \in \mathsf{Mul}(\{\phi_D\})$ and $\mathcal{F} \notin \mathsf{Mul}(\{\psi\})$, because ϕ_D does not have any multimorphisms except for multi-projections.

Example 3.5.3. An example of the cost function from Theorem 3.5.2 is the following function defined on $D = \{1, 2, 3\}$ as follows:

$$\phi(x,y) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = 1 \text{ and } y = 2, \\ 1 & \text{if } x = 2 \text{ and } y = 1, \\ 17 & \text{if } (x = 1 \text{ and } y = 3) \text{ or } (x = 3 \text{ and } y = 1), \\ 19 & \text{if } (x = 2 \text{ and } y = 3) \text{ or } (x = 3 \text{ and } y = 2), \\ 20 & \text{if x=y.} \end{cases}$$

[3]Using the alternative definition of fractional polymorphisms from Remark 2.4.14, these would be *fractional projections*, see [CCJ06] for more details.

Martin Cooper has proved that the family of cost functions used in the proof of Theorem 3.5.2 is NP-hard. More formally, for every ϕ_D used in the proof of Theorem 3.5.2, the valued constraint language $\Gamma = \{\phi_D\}$ is intractable.

Given a valued constraint language Γ, Theorem 3.5.1 shows that if Γ does not have non-trivial fractional polymorphisms, then Γ is intractable. Above-mentioned results of Martin Cooper show that having a fractional polymorphism does not imply tractability of a valued constraint language. We conjecture, on the other hand, that having a multimorphism implies tractability of a valued constraint language.

Conjecture 3.5.4. *A valued constraint language is tractable if, and only if, it has a non-trivial multimorphism.*

Remark 3.5.5. One direction of Conjecture 3.5.4 is very strong and difficult to prove: showing that having a multimorphism implies tractability would imply the Dichotomy Conjecture of Feder and Vardi [FV98], mentioned in Remark 2.2.13.

However, we believe that the other direction is not as difficult. One needs to show that if a valued constraint language Γ does not have any non-trivial multimorphisms, then Γ is intractable. This would mean that multimorphisms are a necessary condition for tractability of valued constraints similarly to the role of polymorphisms for tractability of crisp constraints [JCG97].

3.6 Summary

We have investigated the expressive power of crisp and valued constraints. We have presented a construction to determine whether a given cost function is expressible over a given valued constraint language. We have also presented a construction to determine whether a given fractional polymorphism belongs to a fractional clone. We have linked questions regarding the expressive power of valued constraint to linear programming.

Related work Schnoor and Schnoor have considered Galois connections for various variants of the CSP [SS06].

Open problems There are several interesting open questions in this area. First, are there any other algebraic operations, apart from fractional polymorphisms and feasibility polymorphisms, that characterise the expressive power of valued constraints? Second, what is the structure of fractional clones and what is the closure operator? Next, do multimorphisms alone characterise the tractability of valued constraints? (In other words, is Conjecture 3.5.4 true?) Or at least, are multimorphisms a necessary condition for tractability?

Unary polymorphisms play an important role in the complexity of CSPs: they can reduce the domain (so-called *squashing*) of the variables. It would be interesting to find out whether unary fractional polymorphisms can be useful in a similar way.

Recall from Example 2.2.18 the language Γ_{fix} of constant constraints, which are unary relations forcing their argument to take a fixed value. In the CSP, it has been

shown that adding constant constraints to a tractable language which is a core does not change the complexity; that is, $\mathsf{CSP}(\Gamma)$ is linear-time equivalent to $\mathsf{CSP}(\Gamma \cup \Gamma_{\mathsf{fix}})$, provided Γ is a core [BKJ05]. It is an open question whether this is true for valued constraints. However, our recent results show that it is unlikely that $\mathsf{VCSP}(\Gamma)$ and $\mathsf{VCSP}(\Gamma \cup \Gamma_{\mathsf{fix}})$ are linear-time equivalent [ŽJ09b].

CHAPTER 4

Expressive Power of Fixed-Arity Languages

I just wondered how things were put together.
Claude Shannon (1916–2001)

This chapter is based on the following papers:

[CJŽ08] D.A. Cohen, P.G. Jeavons, and S. Živný. The Expressive Power of Valued Constraints: Hierarchies and Collapses. *Theoretical Computer Science*, 409(1):137–153, 2008.
Earlier version in *Proceedings of the 13th International Conference on Principles and Practice of Contraint Programming (CP'07)*, volume 4741 of *Lecture Notes in Computer Science*, pages 798–805. Springer, 2007.

[ZŽ09] B. Zanuttini and S. Živný. A Note on Some Collapse Results of Valued Constraints. *Information Processing Letters*, 109(11):534–538, 2009.

4.1 Introduction

In this chapter, we present our results on the expressive power of various classes of valued constraints. Most of the results are of the following form: let $\mathcal{C}$ be a class of valued constraints with cost functions of unbounded arities, then $\mathcal{C}$ can be expressed by a subset of $\mathcal{C}$ consisting of valued constraints with cost functions of a fixed bounded arity. The only known class for which this is not true is the class of finite-valued max-closed cost functions of different arities.

This chapter is organised as follows. In Section 4.2, we define various classes of cost functions, and present our results. In Sections 4.3, 4.4, and 4.5, we present our results for *crisp*, *finite-valued*, and *general* cost functions, respectively. We present both algebraic proofs of most results, which have been published in [CJŽ08], and

also several alternative, non-algebraic proofs of some of the results, which have been published in [ZŽ09]. Finally, in Section 4.6, we present some more results on the algebraic properties of finite-valued max-closed cost functions.

4.2 Results

First, we present our results on the expressive power of valued constraint languages containing all cost functions up to some fixed arity over some fixed domain.

Recall that general cost functions can take both finite and infinite costs.

Definition 4.2.1. For every $d \geq 2$ we define the following:

- $\mathbf{R}_{d,m}$ denotes the set of all relations of arity at most m over a domain of size d, and $\mathbf{R}_d \stackrel{\text{def}}{=} \cup_{m\geq 0}\mathbf{R}_{d,m}$.
- $\mathbf{F}_{d,m}$ denotes the set of all finite-valued cost functions of arity at most m over a domain of size d, and $\mathbf{F}_d \stackrel{\text{def}}{=} \cup_{m\geq 0}\mathbf{F}_{d,m}$.
- $\mathbf{G}_{d,m}$ denotes the set of all general cost functions of arity at most m over a domain of size d, and $\mathbf{G}_d \stackrel{\text{def}}{=} \cup_{m\geq 0}\mathbf{G}_{d,m}$.

We will prove the following Theorem showing that crisp cost functions of a fixed arity can express crisp cost functions of arbitrary arities, and same holds for both finite-valued and general cost functions.

Theorem 4.2.2. *For all $d \geq 3$ and $f \geq 2$:*

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

We will then consider important subsets of these languages defined for totally-ordered domains, containing the so-called *max-closed* cost functions, which are defined below.

Recall that the function Max denotes the standard binary function which returns the larger of its two arguments.

Definition 4.2.3. A cost function ϕ is called *max-closed* if $\{\langle 2, \textsc{Max}\rangle\} \in \mathsf{fPol}(\{\phi\})$.

Observation 4.2.4. Equivalently, ϕ is max-closed if $\langle \textsc{Max}, \textsc{Max}\rangle \in \mathsf{Mul}(\{\phi\})$.

Definition 4.2.5. For every $d \geq 2$ we define the following:

- $\mathbf{R}^{\max}_{d,m}$ denotes the set of all max-closed relations of arity at most m over an ordered domain of size d, and $\mathbf{R}^{\max}_{d} \stackrel{\text{def}}{=} \cup_{m\geq 0}\mathbf{R}^{\max}_{d,m}$.
- $\mathbf{F}^{\max}_{d,m}$ denotes the set of all finite-valued max-closed cost functions of arity at most m over an ordered domain of size d, and $\mathbf{F}^{\max}_{d} \stackrel{\text{def}}{=} \cup_{m\geq 0}\mathbf{F}^{\max}_{d,m}$.
- $\mathbf{G}^{\max}_{d,m}$ denotes the set of all general max-closed cost functions of arity at most m over an ordered domain of size d, and $\mathbf{G}^{\max}_{d} \stackrel{\text{def}}{=} \cup_{m\geq 0}\mathbf{G}^{\max}_{d,m}$.

We will show below that the following Theorem holds for these sets of max-closed cost functions. Note that the result establishes an infinite hierarchy for finite-valued max-closed cost functions.

Theorem 4.2.6. *For all $d \geq 3$ and $f \geq 2$:*

1. $\langle \mathbf{R}^{\max}_{2,1}\rangle \subsetneq \langle \mathbf{R}^{\max}_{2,2}\rangle \subsetneq \langle \mathbf{R}^{\max}_{2,3}\rangle = \mathbf{R}^{\max}_{2}$.
2. $\langle \mathbf{R}^{\max}_{d,1}\rangle \subsetneq \langle \mathbf{R}^{\max}_{d,2}\rangle = \mathbf{R}^{\max}_{d}$.
3. $\langle \mathbf{F}^{\max}_{f,1}\rangle \subsetneq \langle \mathbf{F}^{\max}_{f,2}\rangle \subsetneq \langle \mathbf{F}^{\max}_{f,3}\rangle \subsetneq \langle \mathbf{F}^{\max}_{f,4}\rangle \cdots$
4. $\langle \mathbf{G}^{\max}_{2,1}\rangle \subsetneq \langle \mathbf{G}^{\max}_{2,2}\rangle \subsetneq \langle \mathbf{G}^{\max}_{2,3}\rangle = \mathbf{G}^{\max}_{2}$.
5. $\langle \mathbf{G}^{\max}_{d,1}\rangle \subsetneq \langle \mathbf{G}^{\max}_{d,2}\rangle = \mathbf{G}^{\max}_{d}$.

In the rest of this chapter, we will prove Theorem 4.2.2 and Theorem 4.2.6. For some results, we present both an algebraic and a non-algebraic proof. Quite often the algebraic proofs are more involved. However, these proofs provide us with more than a statement of the result; they show us the structure of the algebraic properties of the corresponding class of cost functions. Moreover, for the separating result in Theorem 4.2.6 (3), the algebraic properties play a crucial role. In Section 4.5, we characterise the fractional clone of general max-closed cost functions. In Section 4.6, we characterise the multi-clone and fractional clone of finite-valued max-closed cost functions.

4.3 The expressive power of arbitrary relations and max-closed relations

In this section, we consider the expressive power of valued constraint languages containing only *crisp* cost functions, that is, *relations*.

We consider the languages containing all relations up to some fixed arity over some fixed domain, and the languages containing all *max-closed* relations up to some fixed arity over some fixed totally-ordered domain. In both cases, we show that the relations of a fixed arity can express all relations of arbitrary arities.

The class of *crisp* max-closed cost functions has been first introduced (as a class of relations) in [JC95] and shown to be tractable. In other words, $\mathsf{VCSP}(\Gamma)$ is known

to be polynomial-time solvable for any set Γ consisting of max-closed relations over any finite set D. A number of examples of max-closed relations are given in [JC95].

Remark 4.3.1. The max-closed property generalises the *X-underbar* property studied in the context of graph homomorphisms [HN04], which applies only to binary relations.

It is well known that any relation can be expressed as a propositional formula in Conjunctive Normal Form (CNF), simply as a conjunction of clauses which disallow tuples not in the relation. Hence we have the following characterisation of $\mathbf{R}_{d,m}$.

Proposition 4.3.2. *A relation $R \in \mathbf{R}_{d,m}$ if, and only if, there is some formula ψ such that $\langle v_1, \ldots, v_m \rangle \in R \Leftrightarrow \psi(v_1, \ldots, v_m)$ and ψ is a conjunction of clauses of the form $(v_1 \neq a_1) \vee \ldots \vee (v_m \neq a_m)$ for some constants $a_1, \ldots, a_m$.*

We also have a similar characterisation for $\mathbf{R}^{\max}_{d,m}$, adapted from Theorem 5.2 of [JC95]. (The same result has also been obtained in [GHSZ08].)

Theorem 4.3.3 ([JC95])**.** *A relation $R \in \mathbf{R}^{\max}_{d,m}$ if, and only if, there is some formula ψ such that $\langle v_1, \ldots, v_m \rangle \in R \Leftrightarrow \psi(v_1, \ldots, v_m)$ and ψ is a conjunction of clauses of the form $(v_1 > a_1) \vee \ldots \vee (v_m > a_m) \vee (v_i < b_i)$ for some constants $a_1, \ldots, a_m, b_i$.*

Note that in the special case of a Boolean domain (that is, when $d = 2$) this restricted form of clauses is equivalent to a disjunction of literals with at most one negated literal; clauses of this form are sometimes called *anti-Horn* clauses.

It is well known that for every $d \geq 2$, $\mathsf{Pol}(\mathbf{R}_d)$ is equal to the set of all possible projection operations [DW02]. We now characterise the polymorphisms of $\mathbf{R}^{\max}_d$.

Definition 4.3.4. Let D be a fixed totally-ordered set.

- The k-ary function on D which returns the largest of its k arguments in the given ordering of D is denoted $\textsc{Max}_k$.
- The k-ary function on D which returns the smallest of its k arguments in the given ordering of D is denoted $\textsc{Min}_k$.
- The k-ary function on D which returns the second largest of its $k \geq 2$ arguments in the given ordering of D is denoted $\textsc{Second}_k$.

The function $\textsc{Max}_2$ will be denoted $\textsc{Max}$ and the function $\textsc{Min}_2$ will be denoted $\textsc{Min}$.

Definition 4.3.5. Let $I = \{i_1, \ldots, i_n\} \subseteq \{1, \ldots, k\}$ be a set of indices. Define the k-ary function

$$\textsc{Max}_I(x_1, \ldots, x_k) \stackrel{\text{def}}{=} \textsc{Max}_n(x_{i_1}, \ldots, x_{i_n}).$$

For every k, there are exactly $2^k - 1$ functions of the form $\textsc{Max}_I$ for $\emptyset \neq I \subseteq \{1, \ldots, k\}$.

Proposition 4.3.6. *For all $d \geq 2$,*

$$\mathsf{Pol}(\mathbf{R}_d^{\max}) = \{\textsc{Max}_I \mid \emptyset \neq I \subseteq \{1, \ldots, k\}, k = 1, 2, \ldots\}.$$

Proof. When $|I| = 1$, the corresponding function $\textsc{Max}_I$ is just a projection operation, and every projection is a polymorphism of every relation (see Observation 2.4.4).

If $\textsc{Max} \in \mathsf{Pol}(\{R\})$, then $\textsc{Max}_I \in \mathsf{Pol}(\{R\})$ for every $\emptyset \neq I \subseteq \{1, \ldots, k\}$. This is because $\mathsf{Pol}(\{R\})$ is closed under function composition and contains all projection operations, and every $\textsc{Max}_I$ can be obtained by function composition from the function Max and the projection operations.

We now prove that the operations of the form $\textsc{Max}_I$ are the *only* polymorphisms of $\mathbf{R}_d^{\max}$. Suppose, for contradiction, that f is a k-ary polymorphism of $\mathbf{R}_d^{\max}$ which is different from $\textsc{Max}_I$ for every $\emptyset \neq I \subseteq \{1, \ldots, k\}$. It follows that, for each I such that $\emptyset \neq I \subseteq \{1, \ldots, k\}$, there is a k-tuple t_I, such that $f(t_I) \neq \textsc{Max}_I(t_I)$. Let n be the total number of different tuples t_I, that is, $n = |\{t_I \mid \emptyset \neq I \subseteq \{1, \ldots, k\}\}| \leq 2^k - 1$ and denote these tuples by $t_1, \ldots, t_n$. Now consider the n-ary relation $R = \{\langle t_1[j], \ldots, t_n[j]\rangle\}_{1 \leq j \leq k}$. Define $R_0 = R$ and $R_{i+1} = R_i \cup \{\textsc{Max}(u, v) \mid u, v \in R_i\}$ for every $i \geq 0$. Clearly, $R_i \subseteq R_{i+1}$ and since there is only a finite number of different n-tuples, there is an l such that $R_l = R_{l+i}$ for every $i \geq 0$. Define R' to be the closure of R under Max, that is, $R' = R_l$. Clearly, R' is max-closed and every tuple t of R' is of the form $t = \textsc{Max}_j(u_{i_1}, \ldots, u_{i_j})$ for some $j \geq 1$ and $u_{i_1}, \ldots, u_{i_j} \in R$. We have constructed R so that the application of f to the tuples of R results in a tuple t which is different from every tuple of this form, and hence $t \notin R'$. Therefore, $f \notin \mathsf{Pol}(\{R'\})$, which means that $f \notin \mathsf{Pol}(\mathbf{R}_d^{\max})$. □

We now consider the expressive power of $\mathbf{R}_{d,m}$ and $\mathbf{R}_{d,m}^{\max}$.

It is clear that binary relations have greater expressive power than unary relations, so our first result is not unexpected, but it provides a simple illustration of the use of the algebraic approach.

Proposition 4.3.7. *For all $d \geq 2$, $\langle \mathbf{R}_{d,1} \rangle \subsetneq \langle \mathbf{R}_{d,2} \rangle$ and $\langle \mathbf{R}_{d,1}^{\max} \rangle \subsetneq \langle \mathbf{R}_{d,2}^{\max} \rangle$.*

Proof. Notice for example that $\textsc{Min} \in \mathsf{Pol}(\mathbf{R}_{d,1})$ and consequently $\textsc{Min} \in \mathsf{Pol}(\mathbf{R}_{d,1}^{\max})$ but $\textsc{Min} \notin \mathsf{Pol}(\mathbf{R}_{d,2})$ and $\textsc{Min} \notin \mathsf{Pol}(\mathbf{R}_{d,2}^{\max})$. The result then follows from Theorem 2.4.6. □

4.3.1 Relations over a Boolean domain

As a first step, we now focus on the special case of relations over a Boolean domain, that is, the case when $d = 2$. This special case has been studied in detail in [BRSV05]. Here, we give a brief independent derivation of the relevant results using the techniques introduced above. We first show that the set of all ternary relations over a Boolean domain has fewer polymorphisms than the set of all binary relations, and hence has a greater expressive power. We also establish similar results for max-closed relations over a Boolean domain.

Proposition 4.3.8. MAJORITY $\in \mathsf{Pol}(\mathbf{R}_{2,2})$ *and* MAJORITY $\in \mathsf{Pol}(\mathbf{R}^{\max}_{2,2})$*, where* MAJORITY *is the unique ternary function on a 2-element set which returns the argument value that occurs most often.*

Proof. Let R be an arbitrary binary Boolean relation. Let $a = \langle a_1, a_2\rangle$, $b = \langle b_1, b_2\rangle$ and $c = \langle c_1, c_2\rangle$ be three pairs belonging to R. Note that since the domain size is 2, the pair $\langle \textsc{Majority}(a_1, b_1, c_1), \textsc{Majority}(a_2, b_2, c_2)\rangle$ is equal to at least one of a, b, c, and hence belongs to R. □

Proposition 4.3.9. MAJORITY $\notin \mathsf{Pol}(\mathbf{R}_{2,3})$ *and* MAJORITY $\notin \mathsf{Pol}(\mathbf{R}^{\max}_{2,3})$.

Proof. Consider the ternary Boolean max-closed relation R consisting of all triples except $\langle 0, 0, 0\rangle$. To see that MAJORITY is not a polymorphism of R, consider the triples $\langle 0, 0, 1\rangle$, $\langle 0, 1, 0\rangle$ and $\langle 1, 0, 0\rangle$. The application of MAJORITY to these tuples results in the triple $\langle 0, 0, 0\rangle$ which is not in R. □

However, we now show that *ternary* Boolean relations have the same expressive power as *all* Boolean relations. In other words, any Boolean relation of arbitrary arity is expressible by relations of arity at most three. The same result also holds for max-closed Boolean relations.

Proposition 4.3.10. $\mathbf{R}_2 \subseteq \langle \mathbf{R}_{2,3}\rangle$ *and* $\mathbf{R}^{\max}_2 \subseteq \langle \mathbf{R}^{\max}_{2,3}\rangle$.

Proof. By Proposition 4.3.2, any Boolean relation $R \in \mathbf{R}_2$ can be expressed as a CNF formula ψ. By the standard SATISFIABILITY to 3-SATISFIABILITY reduction [GJ79], there is a 3-CNF formula ψ' expressing R such that ψ is satisfiable if, and only if, ψ' is satisfiable.

Since the standard SATISFIABILITY to 3-SATISFIABILITY reduction preserves the anti-Horn form of clauses, the same result holds for max-closed Boolean relations. □

Combining these results with Theorem 2.4.6, we obtain the following result.

Theorem 4.3.11.

1. $\langle \mathbf{R}_{2,1}\rangle \subsetneq \langle \mathbf{R}_{2,2}\rangle \subsetneq \langle \mathbf{R}_{2,3}\rangle = \mathbf{R}_2$.
2. $\langle \mathbf{R}^{\max}_{2,1}\rangle \subsetneq \langle \mathbf{R}^{\max}_{2,2}\rangle \subsetneq \langle \mathbf{R}^{\max}_{2,3}\rangle = \mathbf{R}^{\max}_2$.

4.3.2 Relations over larger domains

For relations over a domain with 3 or more elements, similar results can be obtained. In fact, in this case we show that *any* relation can be expressed using *binary* relations.

Proposition 4.3.12. *For all* $d \geq 3$, $\mathbf{R}_d \subseteq \langle \mathbf{R}_{d,2}\rangle$.

Proof. Without loss of generality, assume that $D = \{0, \ldots, M\}$, where $M = d - 1$. Define the binary relation R_d by

$$R_d \stackrel{\text{def}}{=} \{\langle 0, i\rangle, \langle i, 0\rangle \mid 0 \leq i \leq M\} \cup \{\langle i, i+1\rangle \mid 1 \leq i < M\}.$$

It is known that the only polymorphisms of the relation R_d are projection operations [Fea95]. Hence, by Theorem 2.4.6, $\langle \{R_d\}\rangle = \mathbf{R}_d$. □

We now present a non-algebraic proof.

Proof. (alternative proof of Proposition 4.3.12)

By Proposition 4.3.2, every relation is logically equivalent to some conjunction of clauses. Therefore, Proposition 4.3.10 gives a weaker result that $\mathbf{R}_d \subseteq \langle \mathbf{R}_{d,3} \rangle$. We need to show how to express a clause C of length 3 (over the domain D, where $d = |D| \geq 3$) as a conjunction of clauses of length 2. Let $D = \{1, \ldots, d\}$ and $C = (U_1(x_1) \vee U_2(x_2) \vee U_3(x_3))$ for some literals (unary relations) U_i, $1 \leq i \leq 3$. We claim that C is equivalent to $\exists y C' = (U_1(x_1) \vee N_1(y)) \wedge (U_2(x_2) \vee N_2(y)) \wedge (U_3(x_3) \vee N_3(y))$ where y is a new variable and $N_1(y) = D \setminus \{1\}$ ("not 1"), $N_2(y) = D \setminus \{2\}$ and $N_3(y) = \{1, 2\}$. It is not difficult to see that a satisfying assignment of C can be extended to a satisfying assignment of C' and conversely, a satisfying assignment of C' gives a satisfying assignment of C. □

By investigating the polymorphisms of binary max-closed relations, we now show that max-closed relations over non-Boolean domains can also be expressed using binary relations.

Theorem 4.3.13. *For all $d \geq 3$, $\mathbf{R}_d^{\max} \subseteq \langle \mathbf{R}_{d,2}^{\max} \rangle$.*

Proof. We will show that $\mathsf{Pol}(\mathbf{R}_{d,2}^{\max}) \subseteq \mathsf{Pol}(\mathbf{R}_d^{\max})$. The result then follows from Theorem 2.4.6.

Without loss of generality, assume that $D = \{0, \ldots, M\}$ where $M = d - 1$. Let $f \in \mathsf{Pol}(\mathbf{R}_{d,2}^{\max})$ be an arbitrary k-ary polymorphism. By Proposition 4.3.6, it is enough to show that $f = \textsc{Max}_I$ for some $\emptyset \neq I \subseteq \{1, \ldots, k\}$.

First note that for any subset $S \subseteq D$, the binary relation $R = \{\langle a, a \rangle \mid a \in S\}$ is max-closed, so $f(x_1, \ldots, x_k) \in \{x_1, \ldots, x_k\}$. In other words, f is conservative.

If $f = \textsc{Max}_{\{1,\ldots,k\}}$ we are done. Otherwise, there exist $a_1, \ldots, a_k \in D$ such that $a_i = \textsc{Max}_k(a_1, \ldots, a_k)$ and $a_i > f(a_1, a_2, \ldots, a_k) = a_j$. Without loss of generality, in order to simplify our notation, assume that $i = 1$ and $j = 2$, that is, $a_1 = \textsc{Max}_k(a_1, \ldots, a_k)$ and $a_1 > f(a_1, a_2, \ldots, a_k) = a_2$. We will show that f does not depend on its first parameter.

For any fixed $x_2, \ldots, x_k \in D$, we denote the tuple $\langle x_2, \ldots, x_k \rangle$ by $\bar{x}$, and we define the binary max-closed relation

$$R_{\bar{x}} \stackrel{\text{def}}{=} (\{a_2, \ldots, a_k\} \times \{x_2, \ldots, x_k\}) \cup (\{a_1\} \times D).$$

Now consider the function $g_{\bar{x}}(r) = f(r, x_2, \ldots, x_k)$. Note that $g_{\bar{x}}(r)$ is a restriction of f with all arguments except the first one fixed.

Claim 1. $\forall r \in D,\ g_{\bar{x}}(r) \in \{x_2, \ldots, x_k\}$.

To establish this claim, note that for all $r \in D$ we have $\langle a_1, r \rangle \in R_{\bar{x}}$, and $\{\langle a_j, x_j \rangle \mid j = 2, \ldots, k\} \subseteq R_{\bar{x}}$. Since f is a polymorphism of $R_{\bar{x}}$ and $f(a_1, a_2, \ldots, a_k) = a_2$, it follows from the definition of $R_{\bar{x}}$ that $g_{\bar{x}}(r) \in \{x_2, \ldots, x_k\}$.

Now we show that if the largest element of the domain, M, is not among $x_2, \ldots, x_k$, then $g_{\bar{x}}(r)$ is constant.

Claim 2. $M \notin \{x_2, \ldots, x_k\} \Rightarrow \forall r \in D,\ g_{\bar{x}}(r) = g_{\bar{x}}(M)$.

To establish this claim, define the binary max-closed relation

$$R'_{\bar{x}} \stackrel{\text{def}}{=} (\{M\} \times D) \cup \{\langle x_j, x_j \rangle \mid j = 2, \ldots, k\}.$$

For all $r \in D$ we have $\langle M, r \rangle \in R'_{\bar{x}}$ and $\{\langle x_j, x_j \rangle \mid j = 2, \ldots, k\} \subseteq R'_{\bar{x}}$. By Claim 1, $g_{\bar{x}}(M) = x_i$ for some $2 \leq i \leq k$. Since f is a polymorphism of $R'_{\bar{x}}$, it follows from the definition of $R_{\bar{x}}$ that $g_{\bar{x}}(r) = x_i = g_{\bar{x}}(M)$ for every $r \in D$.

Next we generalise Claim 2 to show that $g_{\bar{x}}(r)$ is constant whenever $x_2, \ldots, x_k$ does not contain all elements of the domain D.

Claim 3. $\{x_2, \ldots, x_k\} \neq D \Rightarrow \forall r \in D,\ g_{\bar{x}}(r) = g_{\bar{x}}(M)$.

To establish this claim, we will show that for every $p \in D$, if $p \notin \{x_2, \ldots, x_k\}$, then $g_{\bar{x}}(r) = g_{\bar{x}}(M)$ for every $r \in D$. Note that the case $p = M$ is already proved by Claim 2. For any $p \in D \setminus \{M\}$, define the binary max-closed relation $R_p = \{\langle d, \Delta_p(d) \rangle \mid d \in D\}$, where

$$\Delta_p(x) \stackrel{\text{def}}{=} \begin{cases} x & \text{if } x \leq p, \\ x - 1 & \text{if } x > p. \end{cases}$$

For all $r \in D$ we have $\langle r, \Delta_p(r) \rangle \in R_p$ and $\{\langle x_j, \Delta_p(x_j) \rangle \mid j = 2, \ldots, k\} \subseteq R_p$. Since f is a polymorphism of R_p, it follows from the definition of R_p that for every $r \in D$, $g_{\bar{x}}(r) \in \Delta_p^{-1}(g_{\Delta_p(\bar{x})}(\Delta_p(r)))$.

Since $M \notin \{\Delta_p(d) \mid d \in D\}$, we know, by Claim 2, that $g_{\Delta_p(\bar{x})}(\Delta_p(r))$ is constant. Say $g_{\Delta_p(\bar{x})}(\Delta_p(r)) = k_p$. If $k_p \neq p$, then $|\Delta_p^{-1}(k_p)| = 1$ and so $g_{\bar{x}}$ is constant. Alternatively, if $k_p = p$, then $\Delta_p^{-1}(k_p) = \{p, p+1\}$. In this case if $p \notin \{x_2, \ldots, x_k\}$, then we know, by Claim 1, that $g_{\bar{x}}(r) \neq p$, so $g_{\bar{x}}$ is again constant. This completes the proof of Claim 3.

Claim 4. $g_{\bar{x}}(r)$ is constant.

To establish this claim, define the binary max-closed relations $R_+ = \{\langle d, \Delta_+(d) \rangle \mid d \in D\}$ and $R_- = \{\langle d, \Delta_-(d) \rangle \mid d \in D\}$, where

$$\Delta_+(x) \stackrel{\text{def}}{=} \begin{cases} x & \text{if } x \neq M, \\ x - 1 & \text{if } x = M \end{cases}$$

and

$$\Delta_-(x) \stackrel{\text{def}}{=} \begin{cases} x & \text{if } x \neq 0, \\ x + 1 & \text{if } x = 0. \end{cases}$$

Define $\bar{y} = \langle \Delta_+(x_2), \ldots, \Delta_+(x_k) \rangle$ and $\bar{z} = \langle \Delta_-(x_2), \ldots, \Delta_-(x_k) \rangle$. Since $M \notin \{\Delta_+(d) \mid d \in D\}$ and $0 \notin \{\Delta_-(d) \mid d \in D\}$, we know, by Claim 3, that $g_{\bar{y}}$ and $g_{\bar{z}}$ are both constant.

For every $r \in D$, $\langle r, \Delta_+(r) \rangle \in R_+$ and for every $i = 2, \ldots, k$, $\langle x_i, \Delta_+(x_i) \rangle \in R_+$. Since f is a polymorphism of R_+, and $g_{\bar{y}}$ is constant, $g_{\bar{x}}$ is either constant or for every

$r \in D$, $g_{\bar{x}}(r) \in \{M, M-1\}$. Similarly, for every $r \in D$, $\langle r, \Delta_-(r)\rangle \in R_-$ and for every $i = 2, \ldots, k$, $\langle x_i, \Delta_-(x_i)\rangle \in R_-$. Since f is a polymorphism of R_-, and $g_{\bar{z}}$ is constant, $g_{\bar{x}}$ is either constant or for every $r \in D$, $g_{\bar{x}}(r) \in \{0, 1\}$. Since $|D| > 2$ we know[1] that $|\{M, M-1\} \cap \{0, 1\}| \leq 1$. Hence, in all cases $g_{\bar{x}}$ is constant.

We have shown that if $a_1 = \max(a_1, \ldots, a_k)$ and $f(a_1, \ldots, a_k) < a_1$, then f does not depend on its first parameter. Similarly, by repeating the same argument, we can show that if $f \neq \textsc{Max}_{\{2,\ldots,k\}}$, then f does not depend on its i-th parameter for some i such that $2 \leq i \leq k$. Moreover, further repeating the same argument shows that if f does not depend on any parameter outside of $I \subseteq \{1, \ldots, k\}$ and $f \neq \textsc{Max}_I$, then f does not depend on any of the parameters whose index is in I.

Therefore, either there is some set $I \subseteq \{1, \ldots, k\}$ for which $f = \textsc{Max}_I$ or else f is constant. However, since f is conservative, it cannot be constant. □

Next we present a non-algebraic proof.

Proof. (alternative proof of Theorem 4.3.13)

It is enough to show that any clause of the form $(x_1 \geq a_1 \vee \cdots \vee x_k \geq a_k)$ or $(x_1 \geq a_1 \vee \cdots \vee x_k \geq a_k \vee y \leq b)$, where $a_1, \ldots, a_k, b$ are domain values, can be expressed by a conjunction of anti-Horn clauses over at most two variables.

Let C be a clause in $\mathbf{R}_d^{\max}$:

$$C = (x_1 \geq a_1 \vee x_2 \geq a_2 \vee \cdots \vee x_k \geq a_k \vee y \leq b)$$

(the case without the y literal is even easier and can be handled similarly).

For all $i = 1, \ldots, k-1$, let y_i be a fresh variable. As $d \geq 3$, the domain of each y_i contains at least 3 different values, say $1, 2$ and 3 with the natural order. We define the following conjunction of clauses ψ, where $y_i \in \{1, 3\}$ is used as a shorthand for $y_i \geq 3 \vee y_i \leq 1$ (possible values less than 1 or greater than 3 do not matter) as follows:

$$\begin{array}{rcl} \psi & \stackrel{\text{def}}{=} & (x_1 \geq a_1 \vee y_1 \in \{1,3\}) \ \wedge \ (y_1 \leq 2 \vee x_2 \geq a_2) \wedge \\ & & \bigwedge_{i=2}^{k-1} \left(\begin{array}{l} (y_{i-1} \geq 2 \vee y_i \in \{1,3\}) \\ \wedge \ (y_i \leq 2 \vee x_{i+1} \geq a_{i+1}) \end{array} \right) \wedge \\ & & \wedge \ (y_{k-1} \geq 2 \vee y \leq b). \end{array}$$

The intuition is given by reading the second clause as $(y_1 \geq 3 \rightarrow x_2 \geq a_2)$ and the third one as $(y_1 \leq 1 \rightarrow y_2 \in \{1, 3\})$. Since the first clause reads "either $x_1 \geq a_1$ or $y_1 \geq 3$ or $y_1 \leq 1$", together with the above implications this gives "either $x_1 \geq a_1$ or $x_2 \geq a_2$ or $y_2 \in \{1, 3\}$". Iterating this reasoning, one can see intuitively why the construction works.

More formally, we show that C is logically equivalent to $\exists y_1 \ldots y_{k-1} \psi$. First, let t be a tuple satisfying ψ. Then if t satisfies $x_1 \geq a_1$, we are done. Otherwise, because of the first clause in ψ, t must satisfy (1) $y_1 \geq 3$ or (2) $y_1 \leq 1$. In case (1), because of the second clause in ψ, t must satisfy $x_2 \geq a_2$ and we are done. In case (2), because of the third clause in ψ, t must satisfy $y_2 \geq 3 \vee y_2 \leq 1$, and we proceed by induction.

[1] This is the only place where we use the condition that $|D| \geq 3$.

Conversely, let t be a tuple satisfying C. We show that t can be completed into a model of ψ by assignments to the y_i's.

Assume first that t satisfies $x_1 \geq a_1$. Then completing t with $t(y_i) = 2$ for all $i = 1, \ldots, k-1$ yields a model of ψ whatever the values assigned by t to $x_2, \ldots, x_k, y$. This can be seen by examining each clause in ψ. Now assume that t satisfies $x_{i_0} \geq a_{i_0}$ for some $i_0 \in \{2, \ldots, k\}$. Then completing t with $t(y_i) = 1$ for all $i = 1, \ldots, i_0 - 2$, $t(y_{i_0-1}) = 3$ and $t(y_i) = 2$ for all $i = i_0, \ldots, k-1$ again yields a model of ψ. Finally, assume that t satisfies $y \leq b$. Then completing t with $t(y_i) = 1$ for all $i = 1, \ldots, k-1$ yields a model of ψ, which finishes the proof.

Note that this proof makes clear why the same argument does not work for $d = 2$. Indeed, the "hole" in literal $y_i \in \{1, 3\}$ is necessary, since otherwise this literal would be tautologous and thus, so would every second clause be in ψ. And Theorem 4.3.11 indeed shows that in the Boolean case, ternary relations are both sufficient and necessary. □

Combining these results we obtain the following result:

Theorem 4.3.14. *For all $d \geq 3$,*

1. $\langle \mathbf{R}_{d,1} \rangle \subsetneq \langle \mathbf{R}_{d,2} \rangle = \mathbf{R}_d$.
2. $\langle \mathbf{R}^{\max}_{d,1} \rangle \subsetneq \langle \mathbf{R}^{\max}_{d,2} \rangle = \mathbf{R}^{\max}_d$.

Figure 4.1 summarises the results from this section.

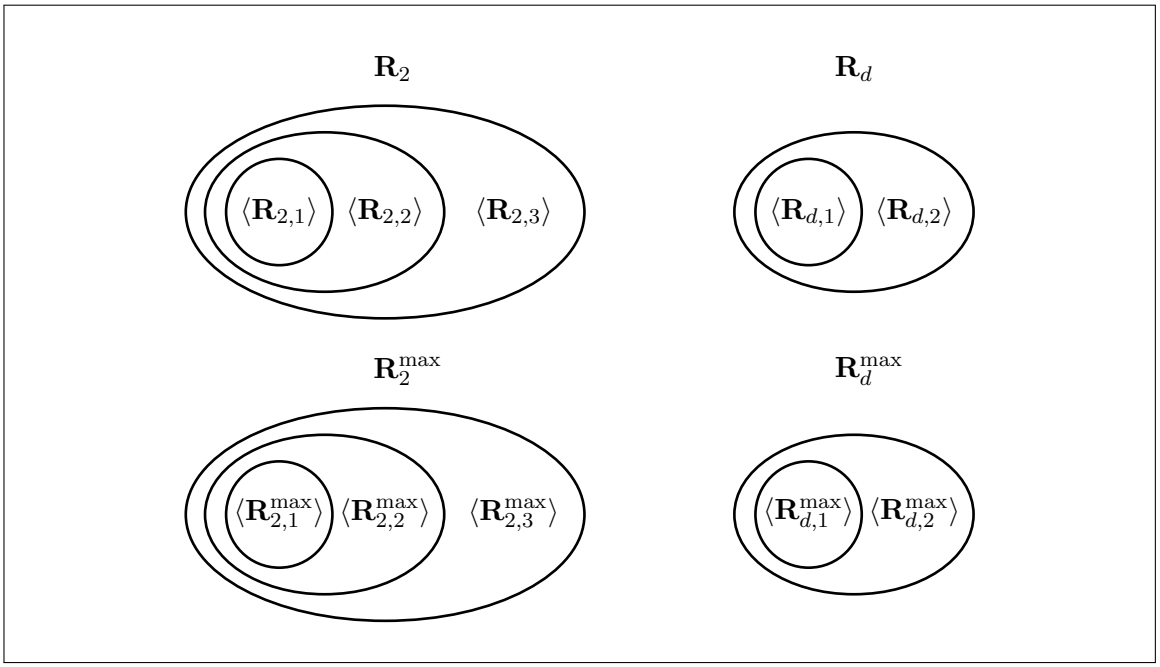

Figure 4.1: Summary of results from Section 4.3, for all $d \geq 3$.

4.4 Finite-valued cost functions

In this section, we consider the expressive power of valued constraint languages containing only *finite-valued* cost functions.

Cost functions from $\mathbf{F}_2$, that is, finite-valued cost functions over a Boolean domain, are also known as *pseudo-Boolean functions* [BH02, CH]. The class of max-closed cost functions is discussed in more detail in [CCJK06] and shown to be tractable. A number of examples of max-closed cost functions are given in [CCJK06].

First we show that the set of all finite-valued cost functions of a certain fixed arity can express all finite-valued cost functions of arbitrary arities. On the other hand, we show that the *max-closed* finite-valued cost functions of any fixed arity *cannot* express all finite-valued max-closed cost functions of any larger arity. Hence we identify an infinite hierarchy of finite-valued cost functions with ever-increasing expressive power.

Proposition 4.4.1. *For all $d \geq 2$, $\langle \mathbf{F}_{d,1} \rangle \subsetneq \langle \mathbf{F}_{d,2} \rangle$ and $\langle \mathbf{F}^{\max}_{d,1} \rangle \subsetneq \langle \mathbf{F}^{\max}_{d,2} \rangle$.*

Proof. Consider the binary weighted mapping $\mathcal{F} = \{\langle 1, \textsc{Min} \rangle, \langle 1, \textsc{Max} \rangle\}$. It is straightforward to verify that $\mathcal{F} \in \mathsf{fPol}(\mathbf{F}_{d,1})$ and $\mathcal{F} \in \mathsf{fPol}(\mathbf{F}^{\max}_{d,1})$.

Now consider the binary finite-valued max-closed cost function ϕ over any domain containing $\{0, 1\}$, defined by $\phi(\langle 0, 0 \rangle) = 1$ and $\phi(\langle ., . \rangle) = 0$ otherwise. Note that ϕ is max-closed but $\mathcal{F}$ is *not* a fractional polymorphism of ϕ. To see this, consider the tuples $\langle 0, 1 \rangle$ and $\langle 1, 0 \rangle$ (see the tableau below).

$$\begin{array}{rcc} & \begin{array}{cc} 0 & 1 \\ 1 & 0 \end{array} & \xrightarrow{\phi} \begin{array}{c} 0 \\ 0 \end{array} \Big\} \sum = 0 \\ \hline \textsc{Min} & \begin{array}{cc} 0 & 0 \end{array} & \xrightarrow{\phi} \begin{array}{c} 1 \\ 0 \end{array} \Big\} \sum = 1 \\ \textsc{Max} & \begin{array}{cc} 1 & 1 \end{array} & \end{array}$$

The result then follows from Theorem 2.4.19. □

Now we prove a collapse result for the set of all finite-valued cost functions over an arbitrary finite domain. This result was previously known for the special case when $d = 2$: as we remarked earlier, any Boolean finite-valued cost function can be represented as a pseudo-Boolean function; using a well-known result from pseudo-Boolean optimisation [BH02, CH], any such function can be expressed using binary pseudo-Boolean functions.

Theorem 4.4.2. *For all $d \geq 2$, $\langle \mathbf{F}_{d,2} \rangle = \mathbf{F}_d$.*

Proof. As mentioned above, the case $d = 2$ follows from well-known results about pseudo-Boolean functions (see Theorem 1 of [BH02]). Let $\phi \in \mathbf{F}_{d,m}$ for some $d \geq 3$ and $m > 2$. We will show how to express ϕ using only unary and binary finite-valued cost functions. Without loss of generality, assume that all cost functions are defined over the set $D = \{0, 1, \ldots, M\}$, where $M = d - 1$, and denote by $D^m = \{t_1, \ldots, t_n\}$ the set of all m-tuples over D. Clearly, $n = d^m$. Let $K \in \mathbb{R}_+$ be a fixed constant such that $K > \max_{t \in D^m} \phi(t)$. For any $e \in D$, let χ^e be the binary finite-valued cost

function defined by

$$\chi^e(x,y) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = e \text{ and } y = 0, \\ 0 & \text{if } x \neq e \text{ and } y = 1, \\ K & \text{otherwise.} \end{cases}$$

For any $r \in \mathbb{R}_+$, let μ^r be the unary finite-valued cost function defined by

$$\mu^r(z) \stackrel{\text{def}}{=} \begin{cases} r & \text{if } z = 0, \\ 0 & \text{otherwise.} \end{cases}$$

We now start building the gadget for ϕ. Let $x_1, \ldots, x_m$ be the variables upon which we wish to construct ϕ, and let $t_i \in D^m$ be an arbitrary fixed tuple. Figure 4.2 shows the part of the gadget for ϕ which ensures that the appropriate cost value is assigned to the tuple of values t_i. The complete gadget for ϕ consists of this part in n copies: one copy on a new set of variables for every $t_i \in D^m$.

Define new variables $y_1^i, \ldots, y_m^i$ and z^i. We apply cost functions on these variables as shown in Figure 4.2. Note that each variable y_j^i, $1 \leq j \leq m$, indicates whether

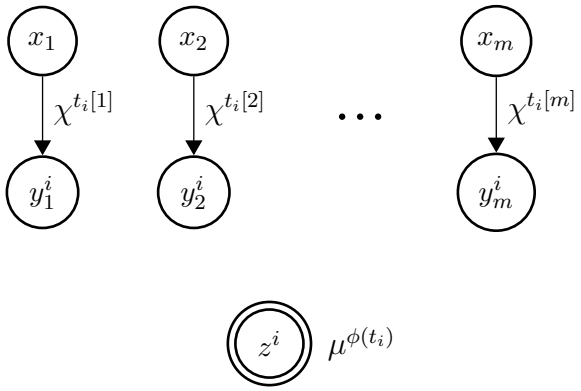

Figure 4.2: A part of the gadget for expressing ϕ (Theorem 4.4.2).

or not x_j is equal to $t_i[j]$: in any minimum-cost assignment, $(y_j^i = 0) \Leftrightarrow (x_j = t_i[j])$. It remains to define the constraints between the variables $y_1^i, \ldots, y_m^i$ and z^i. These will be chosen in such a way that any assignment of the values 0 or 1 to the variables y_j^i can be extended to an assignment to z^i with a total cost equal to the same fixed minimum value. Furthermore, in these extended assignments z^i is assigned 0 if, and only if, all the y_j^i are assigned 0. (We will achieve this by combining appropriate binary finite-valued cost functions over these variables and other fresh variables as described below.) Then, for every possible assignment of values t_i to the variables $x_1, \ldots, x_m$, there is exactly one z^i, $1 \leq i \leq n$, which is assigned the value 0 in any minimum-cost extension of this assignment. The unary constraint with cost function $\mu^{\phi(t_i)}$ on each z^i then ensures that the complete gadget expresses ϕ.

To define the remaining constraints to complete the constraint in Figure 4.2, we define two binary finite-valued cost functions as follows:

$$\phi_1(y,z) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } y = 0 \text{ and } (z = 0 \text{ or } z = 1), \\ 0 & \text{if } y \neq 0 \text{ and } z \neq 0, \\ K & \text{otherwise,} \end{cases}$$

and

$$\phi_2(y,z) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } y = 0 \text{ and } (z = 0 \text{ or } z = 2), \\ 0 & \text{if } y \neq 0 \text{ and } z \neq 0), \\ K & \text{otherwise.} \end{cases}$$

Let $\mathcal{P} = \langle V, D, \mathcal{C} \rangle$ where $V = \{y_1, y_2, z\}$ and $\mathcal{C} = \{\langle\langle y_1, z\rangle, \phi_1\rangle, \langle\langle y_2, z\rangle, \phi_2\rangle\}$. (See Figure 4.3.)

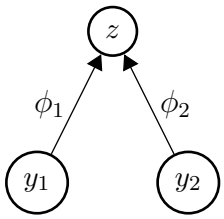

Figure 4.3: $\mathcal{P}$ expressing or_2 over non-Boolean domains (Theorem 4.4.2).

We define or_2 to be the cost function expressed by the gadget $\langle \mathcal{P}, \langle y_1, y_2, z\rangle\rangle$. The cost function $or_2(y_1, y_2, z)$ has the following properties:

- if both y_1, y_2 are assigned the zero value, then the total cost is 0 if, and only if, z is assigned the zero value, otherwise the total cost is either K (if $z = 1$ or $z = 2$) or $2K$ (if $z > 2$);
- if y_1 is assigned the zero value and y_2 a non-zero value, then the total cost is 0 if, and only if, z is assigned 1, otherwise the total cost is K;
- if y_1 is assigned a non-zero value and y_2 the zero value, then the total cost is 0 if, and only if, z is assigned 2, otherwise the total cost is K;
- if both y_1 and y_2 are assigned non-zero values, then the total cost is 0 if, and only if, z is assigned a non-zero value, otherwise the total cost is $2K$.

All these properties of or_2 can be easily verified by examining the so-called *microstructure* [Jég93] of $\mathcal{P}$, as shown in Figure 4.4: this is a graph where the vertices are pairs $\langle v, e\rangle \in V \times D$, and two vertices $\langle v_1, e_1\rangle$ and $\langle v_2, e_2\rangle$ are connected by an edge with weight w if, and only if, there is a valued constraint $\langle\langle v_1, v_2\rangle, c\rangle \in \mathcal{C}$ such that $c(e_1, e_2) = w$. Circles represent particular assignments to particular variables, as indicated in Figure 4.4, and edges are weighted by the cost of the corresponding pair of assignments. Thin edges indicate zero weight, and bold edges indicate weight K.

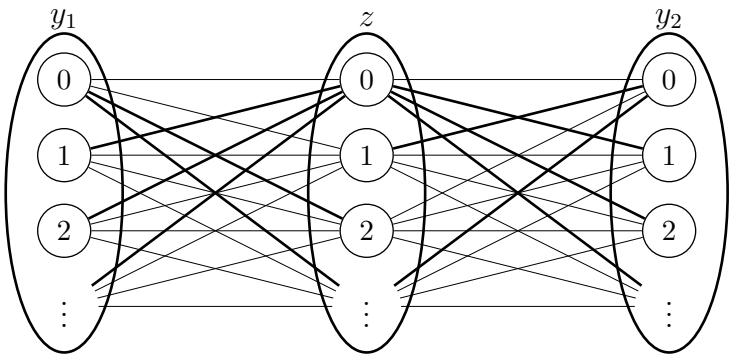

Figure 4.4: Microstructure of the instance $\mathcal{P}$ (Theorem 4.4.2).

We have shown that, in any minimum-cost assignment for $\mathcal{P}$, the variable z takes the value 0 if, and only if, both of the variables y_1 and y_2 take the value 0. Hence the cost function or_2 can be viewed as a kind of 2-input "or-gate", with inputs y_1 and y_2 and output z. By cascading $m-1$ copies of this gadget we can express a cost function $or_m(y_1, \ldots, y_m, z)$, with the following properties:

- if the arguments $y_1, y_2, \ldots, y_m$ are all assigned the zero value, then assigning zero to z gives cost 0, but any non-zero assignment to z gives cost at least K;
- if not all the arguments $y_1, y_2, \ldots, y_m$ are assigned the zero value, then there is a non-zero value $e \in D$ such that assigning e to z gives cost 0, but assigning zero to z gives cost at least K.

Using this combined gadget on the variables $y_1^i, y_2^i, \ldots, y_m^i$ and z^i in Figure 4.2 completes the gadget for ϕ, and hence establishes that $\phi \in \langle \mathbf{F}_{d,2} \rangle$. □

In contrast to this result, the remaining results in this section establish an infinite hierarchy of increasing expressive power for finite-valued *max-closed* cost functions.

Notation 4.4.3. We will say that an m-tuple u *dominates* an m-tuple v, denoted $u \geq v$, if $u[i] \geq v[i]$ for all $1 \leq i \leq m$.

Proposition 4.4.4 ([CCJK06])**.** *An m-ary cost function $\phi : D^m \to \overline{\mathbb{R}}$ is max-closed if, and only if,* $\textsc{Max} \in \mathsf{FPol}(\{\phi\})$ *and ϕ is finitely antitone, that is, for all m-tuples u, v with $\phi(u), \phi(v) < \infty$, $u \leq v \Rightarrow \phi(u) \geq \phi(v)$.*

It follows that the finite-valued max-closed cost functions are simply the finite-valued antitone functions, that is, those functions whose values can only decrease as their arguments get larger. Note that for such functions the expressive power is likely to be rather limited because in any construction the "hidden variables" that are "projected out" can always be assigned the highest values in their domain in order to minimise the cost. Hence, using such hidden variables only adds a constant value to the total cost, and so does not allow more cost functions to be expressed.

We now extend the separation result shown in Proposition 4.4.1 and separate each possible arity.

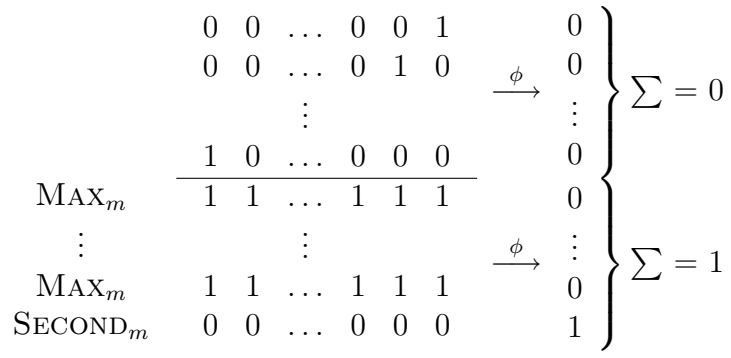

Figure 4.5: $\{\langle m-1, \textsc{Max}_m\rangle, \langle 1, \textsc{Second}_m\rangle\} \notin \mathsf{fPol}(\{\phi\})$ for ϕ (Proposition 4.4.6).

Proposition 4.4.5. *For all $d \geq 2$ and $m \geq 2$, $\{\langle m-1, \textsc{Max}_m\rangle, \langle 1, \textsc{Second}_m\rangle\} \in \mathsf{fPol}(\mathbf{F}^{\max}_{d,m-1})$.*

Proof. Let ϕ be an arbitrary $(m-1)$-ary finite-valued max-closed cost function. Let $t_1, \ldots, t_m$ be $(m-1)$-tuples. We show that there is an i such that the tuple $s = \textsc{Second}_m(t_1, \ldots, t_m)$ dominates t_i, that is, $s[j] \geq t_i[j]$ for $1 \leq j \leq m-1$. To show this we count the number of tuples which can fail to be dominated by s. If a tuple t_p is not dominated by s, for some $1 \leq p \leq m$, it means that there is a position $1 \leq j \leq m-1$ such that $t_p[j] > s[j]$. But since $\textsc{Second}_m$ returns the second biggest value, for every $1 \leq j \leq m-1$, there is at most one tuple which is not dominated by s. Since there are $m \geq 3$ tuples, there must be an i such that t_i is dominated by s. Moreover, $\textsc{Max}_m(t_1, \ldots, t_m)$ clearly dominates all $t_1, \ldots, t_m$. By Proposition 4.4.4, ϕ is antitone and therefore $\{\langle m-1, \textsc{Max}_m\rangle, \langle 1, \textsc{Second}_m\rangle\}$ is a fractional polymorphism of ϕ, by Definition 2.4.10. □

Proposition 4.4.6. *For all $d \geq 2$ and $m \geq 2$, $\{\langle m-1, \textsc{Max}_m\rangle, \langle 1, \textsc{Second}_m\rangle\} \notin \mathsf{fPol}(\mathbf{F}^{\max}_{d,m})$.*

Proof. Let ϕ be the m-ary finite-valued max-closed cost function, over any domain containing $\{0,1\}$, defined by $\phi(\langle 0, \ldots, 0\rangle) = 1$ and $\phi(\langle ., \ldots, .\rangle) = 0$ otherwise. To show that $\{\langle m-1, \textsc{Max}_m\rangle, \langle 1, \textsc{Second}_m\rangle\}$ is *not* a fractional polymorphism of ϕ, consider the m-tuples $\langle 0, \ldots, 0, 1\rangle$, $\langle 0, \ldots, 0, 1, 0\rangle, \ldots, \langle 1, 0, \ldots, 0\rangle$. Each of them is assigned cost 0 by ϕ. But applying the functions $\textsc{Max}_m$ ($(m-1)$ times) and $\textsc{Second}_m$ coordinate-wise results in $m-1$ tuples $\langle 1, \ldots, 1\rangle$, which are assigned cost 0 by ϕ, and one tuple $\langle 0, \ldots, 0\rangle$, which is assigned cost 1 by ϕ (see Figure 4.5). □

Theorem 4.4.7. *For all $d \geq 2$, $\langle \mathbf{F}^{\max}_{d,1}\rangle \subsetneq \langle \mathbf{F}^{\max}_{d,2}\rangle \subsetneq \langle \mathbf{F}^{\max}_{d,3}\rangle \subsetneq \langle \mathbf{F}^{\max}_{d,4}\rangle \cdots$*

Proof. By Propositions 4.4.5 and 4.4.6 and Theorem 2.4.19. □

Figure 4.6 summarises the results from this section.

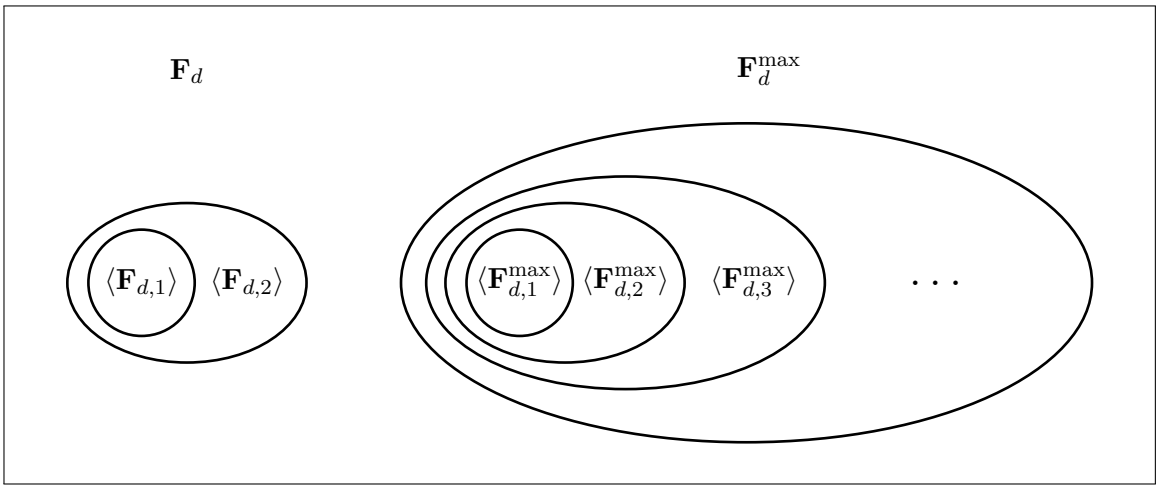

Figure 4.6: Summary of results from Section 4.4, for all $d \geq 2$.

4.5 General cost functions

In this section, we show that general cost functions of a fixed arity can express cost functions of arbitrary arities. Comparing this result with the results of the previous section provides a striking example of the way in which allowing infinite cost values in a valued constraint language can drastically affect the expressibility of cost functions over that language, including finite-valued cost functions.

The class of general max-closed cost functions is known to be tractable [CCJK06].

Once again it is straightforward to establish a separation between unary and binary general cost functions.

Proposition 4.5.1. $\langle \mathbf{G}_{d,1} \rangle \subsetneq \langle \mathbf{G}_{d,2} \rangle$ *and* $\langle \mathbf{G}_{d,1}^{\max} \rangle \subsetneq \langle \mathbf{G}_{d,2}^{\max} \rangle$.

Proof. Identical to the proof of Proposition 4.4.1. □

As with crisp cost functions, in the special case of a Boolean domain, we can show a separation between binary and ternary general cost functions.

Proposition 4.5.2. $\langle \mathbf{G}_{2,2} \rangle \subsetneq \langle \mathbf{G}_{2,3} \rangle$ *and* $\langle \mathbf{G}_{2,2}^{\max} \rangle \subsetneq \langle \mathbf{G}_{2,3}^{\max} \rangle$.

Proof. By Proposition 4.3.8, MAJORITY $\in \mathsf{FPol}(\mathbf{G}_{2,2})$ and MAJORITY $\in \mathsf{FPol}(\mathbf{G}_{2,2}^{\max})$. By Proposition 4.3.9, MAJORITY $\notin \mathsf{FPol}(\mathbf{G}_{2,3})$ and MAJORITY $\notin \mathsf{FPol}(\mathbf{G}_{2,3}^{\max})$. The result then follows by Theorem 2.4.19. □

Next we show a collapse result for general cost functions.

Theorem 4.5.3. *For all* $d \geq 3$, $\langle \mathbf{G}_{d,1} \rangle \subsetneq \langle \mathbf{G}_{d,2} \rangle = \mathbf{G}_d$. *Moreover,* $\langle \mathbf{G}_{2,1} \rangle \subsetneq \langle \mathbf{G}_{2,2} \rangle \subsetneq \langle \mathbf{G}_{2,3} \rangle = \mathbf{G}_2$.

Proof. Let $\phi \in \mathbf{G}_{d,m}$ for some $d \geq 3$ and $m > 2$. It is easy to check that the same construction as in the proof of Theorem 4.4.2 can be used to express ϕ, with $K = \infty$.

Now let $\phi \in \mathbf{G}_{2,m}$ for some $m > 2$. It is easy to check that a similar construction to that used in the proof of Theorem 4.4.2 can be used to express ϕ, where the instance $\mathcal{P}$ is replaced by the ternary Boolean relation which expresses the truth table of a 2-input or-gate. □

Note that the proof shows a slightly stronger result: $\mathbf{G}_2 = \langle \mathbf{R}_{2,3} \cup \mathbf{F}_{2,1} \rangle$, and for all $d \geq 3$, $\mathbf{G}_d = \langle \mathbf{R}_{d,2} \cup \mathbf{F}_{d,1} \rangle$. In other words, all general cost functions can be expressed using *unary* finite-valued cost functions together with ternary relations (in the case $d = 2$), or binary relations (in the case $d \geq 3$).

By investigating feasibility polymorphisms and fractional polymorphisms, we will now show a collapse result for general max-closed cost functions.

First we show that general max-closed cost functions of a fixed arity have the same feasibility polymorphisms as max-closed cost functions of arbitrary arities.

Proposition 4.5.4. *For all $d \geq 3$, $\mathsf{FPol}(\mathbf{G}_{d,2}^{\max}) = \mathsf{FPol}(\mathbf{G}_d^{\max})$. Moreover, $\mathsf{FPol}(\mathbf{G}_{2,3}^{\max}) = \mathsf{FPol}(\mathbf{G}_2^{\max})$.*

Proof. Assume for contradiction, that there is an $f \in \mathsf{FPol}(\mathbf{G}_{d,2}^{\max})$, such that $f \notin \mathsf{FPol}(\mathbf{G}_d^{\max})$. By Definition 4.2.5, $\{\mathsf{Feas}(\phi) \mid \phi \in \mathbf{G}_d^{\max}\} = \mathbf{R}_d^{\max}$. Therefore, such an f would contradict Theorem 4.3.14 since $\mathsf{Pol}(\mathbf{R}_{d,2}^{\max}) = \mathsf{Pol}(\mathbf{R}_d^{\max})$.

Similarly, assume that there is an $f \in \mathsf{FPol}(\mathbf{G}_{2,3}^{\max})$ such that $f \notin \mathsf{FPol}(\mathbf{G}_2^{\max})$. This would contradict Theorem 4.3.11 since $\mathsf{Pol}(\mathbf{R}_{2,3}^{\max}) = \mathsf{Pol}(\mathbf{R}_2^{\max})$. □

We now prove that general max-closed cost functions of a fixed arity have the same fractional polymorphisms as general max-closed cost functions of arbitrary arities. First we characterise the feasibility polymorphisms of general max-closed cost functions.

Proposition 4.5.5. *For all $d \geq 2$,*

$$\mathsf{FPol}(\mathbf{G}_d^{\max}) = \{\textsc{Max}_I \mid \emptyset \neq I \subseteq \{1, \ldots, k\}, k = 1, 2, \ldots\}.$$

Proof. It follows from Definition 4.2.5 that $\{\mathsf{Feas}(\phi) \mid \phi \in \mathbf{G}_d^{\max}\} = \mathbf{R}_d^{\max}$. Therefore, $\mathsf{FPol}(\mathbf{G}_d^{\max}) = \mathsf{FPol}(\mathbf{R}_d^{\max})$ and the result follows from Proposition 4.3.6. □

Next we characterise the fractional polymorphisms of general max-closed cost functions.

Definition 4.5.6. Let $\mathcal{F} = \{(r_1, \textsc{Max}_{S_1}), \ldots, (r_n, \textsc{Max}_{S_n})\}$ be a k-ary weighted mapping and $S \subseteq \{1, \ldots, k\}$.

We define

$$\mathrm{supp}_{\mathcal{F}}\, S \stackrel{\text{def}}{=} \{i \mid S_i \cap S \neq \emptyset\},$$

and

$$\mathrm{wt}_{\mathcal{F}}(S) \stackrel{\text{def}}{=} \sum_{i \in \mathrm{supp}_{\mathcal{F}}(S)} r_i.$$

Theorem 4.5.7. *Let $\mathcal{F} = \{(r_1, \textsc{Max}_{S_1}), \ldots, (r_n, \textsc{Max}_{S_n})\}$ be a k-ary weighted mapping. The following are equivalent:*

1. $\mathcal{F} \in \mathsf{fPol}(\mathbf{G}_d^{\max})$.
2. $\mathcal{F} \in \mathsf{fPol}(\mathbf{G}_{d,1}^{\max})$.
3. *For every subset* $S \subseteq \{1, \ldots, k\}$, $\mathrm{wt}_{\mathcal{F}}(S) \geq |S|$.

Proof. We first show that $\neg(3) \Rightarrow \neg(2) \Rightarrow \neg(1)$.

First suppose that there exists an $S \subseteq \{1, \ldots, k\}$ such that $\mathrm{wt}_{\mathcal{F}}(S) < |S|$. Let $\{a, b\} \subseteq D$ be the two biggest elements of D and $a < b$. Consider the unary cost function ϕ where

$$\phi(x) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = b, \\ 1 & \text{if } x = a, \\ \infty & \text{otherwise.} \end{cases}$$

Certainly $\phi \in \mathbf{G}_{d,1}^{\max}$.

Now let

$$x_i \stackrel{\text{def}}{=} \begin{cases} b & \text{if } i \in S, \\ a & \text{if } i \notin S. \end{cases}$$

We have that

$$\sum_{i=1}^{k} \phi(x_i) = k - |S|, \text{ and}$$

$$\begin{aligned} \sum_{j=1}^{n} r_j \phi(\textsc{Max}_{S_j}(x_1, \ldots, x_k)) &= \sum_{j \notin \mathrm{supp}_{\mathcal{F}} S} r_j \phi(a) + \sum_{j \in \mathrm{supp}_{\mathcal{F}} S} r_j \phi(b) \\ &= k - \mathrm{wt}_{\mathcal{F}}(S) \\ &> k - |S|, \text{ by assumption.} \end{aligned}$$

So $\mathcal{F}$ is not a fractional polymorphism of $\mathbf{G}_{d,1}^{\max}$, and hence not a fractional polymorphism of $\mathbf{G}_d^{\max}$.

To complete the proof we will show that (3) $\Rightarrow$ (1).

Suppose that, for every subset $S \subseteq \{1, \ldots, k\}$, $\mathrm{wt}_{\mathcal{F}}(S) \geq |S|$.

We will first show the existence of a set of non-negative values p_{ji} for $j = 1, \ldots, n$ and $i = 1, \ldots, k$, where

$$\begin{aligned} \sum_{i=1}^{k} p_{ji} &= r_j, \\ \sum_{j=1}^{n} p_{ji} &= 1 \quad \text{and} \\ p_{ji} &= 0 \quad \text{if } i \notin S_j. \end{aligned}$$

Consider the network in Figure 4.7. The capacity from the source to any node x_i is one. The capacity from node y_j to the sink is r_j. There is an arc from node x_i to

node y_j precisely when $i \in S_j$, and the capacity of these arcs is infinite. We show that the flow from x_i to y_j in a maximum flow is the value of p_{ji}

We will use the (s,t)-MIN-CUT MAX-FLOW Theorem to generate the p_{ji}.

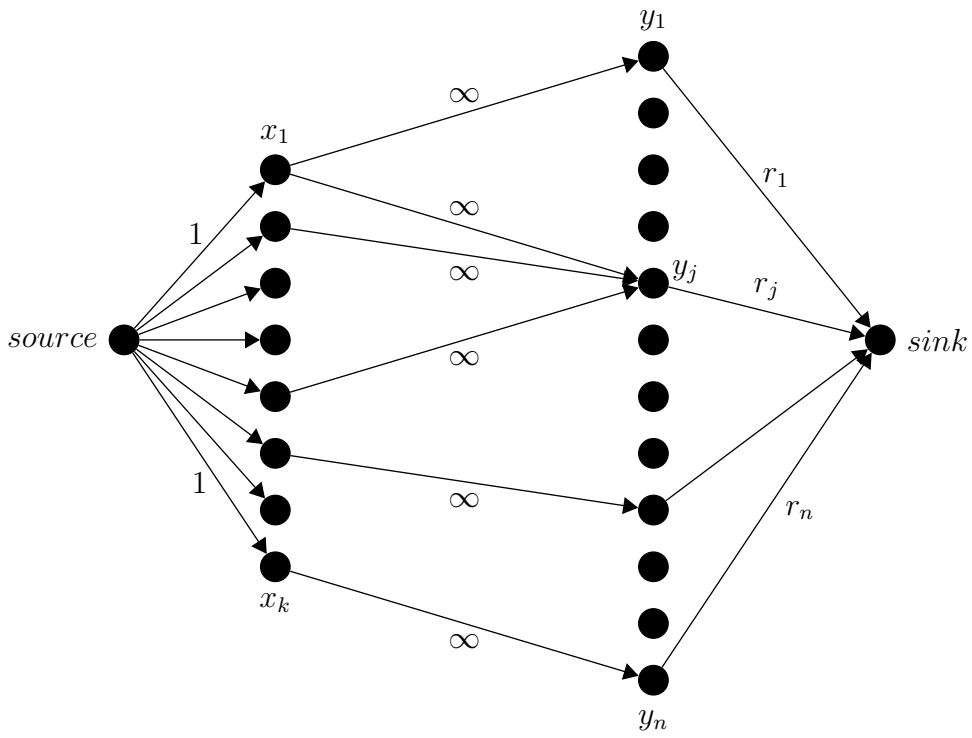

Figure 4.7: Network for p_{ij}'s in the proof of Theorem 4.5.7.

Suppose that we have a minimum cut of this network. Let A be those arcs in this cut from the source to any node x_i. Let $S = \{1, \ldots, k\} - \{i \mid x_i \in A\}$. Since we have a cut we must (at least) cut every arc from the nodes $\{y_j \mid j \in \mathrm{supp}_{\mathcal{F}}(S)\}$ to the sink. By assumption $\mathrm{wt}_{\mathcal{F}}(S) \geq |S|$ and so this cut has total cost at least k. Certainly there is a cut of cost exactly k (cut all arcs from the source), and so the max-flow through this network is precisely k. Such a flow can only be achieved if each arc from the source and each arc to the sink is filled to its capacity. The flow along the arc from x_i to y_j then gives the required value for p_{ji}.

Now we will use these values p_{ji} to show that $\mathcal{F}$ is indeed a fractional polymorphism of $\mathbf{G}_d^{\max}$.

Let $t_1, \ldots, t_k$ be m-ary tuples and $\phi \in \mathbf{G}_{d,m}^{\max}$ be an m-ary cost function. We have to show the following:

$$\sum_{i=1}^{k} \phi(t_i) \geq \sum_{j=1}^{n} r_j \phi(\mathrm{MAX}_{S_j}(t_1, \ldots, t_k)). \tag{4.1}$$

If any $\phi(t_i)$ is infinite, then this inequality clearly holds.

By Proposition 4.5.5, all $\textsc{Max}_{S_j}$, $1 \leq j \leq n$, are feasibility polymorphisms of $\mathbf{G}_d^{\max}$. Therefore, if all $\phi(t_i)$ are finite, then all $\phi(\textsc{Max}_{S_j}(t_1, \ldots, t_k))$ are finite as well.

By definition of p_{ji} and using that $p_{ji} = 0$ whenever $i \notin S_j$ we have that

$$\sum_{j=1}^{n} r_j \phi(\textsc{Max}_{S_j}(t_1, \ldots, t_k)) = \sum_{j=1}^{n} \sum_{i \in S_j} p_{ji} \phi(\textsc{Max}_{S_j}(t_1, \ldots, t_k)).$$

Now, since ϕ is antitone, we have

$$\sum_{j=1}^{n} \sum_{i \in S_j} p_{ji} \phi(\textsc{Max}_{S_j}(t_1, \ldots, t_k)) \leq \sum_{j=1}^{n} \sum_{i \in S_j} p_{ji} \phi(t_i)$$

Since $p_{ji} = 0$ whenever $i \notin S_j$ we have that

$$\sum_{j=1}^{n} \sum_{i \in S_j} p_{ji} \phi(t_i) = \sum_{j=1}^{n} \sum_{i=1}^{k} p_{ji} \phi(t_i)$$

Finally, since $\sum_{j=1}^{n} p_{ji} = 1$ we have established Inequality (4.1). □

Theorem 4.5.8. *For all $d \geq 3$, $\mathsf{fPol}(\mathbf{G}_{d,2}^{\max}) = \mathsf{fPol}(\mathbf{G}_d^{\max})$. Moreover, $\mathsf{fPol}(\mathbf{G}_{2,3}^{\max}) = \mathsf{fPol}(\mathbf{G}_2^{\max})$.*

Proof. By Proposition 4.5.4, $\mathbf{G}_{d,2}^{\max}$ and $\mathbf{G}_d^{\max}$ have the same feasibility polymorphisms. Also, $\mathbf{G}_{2,3}^{\max}$ and $\mathbf{G}_2^{\max}$ have the same feasibility polymorphisms. By Proposition 4.5.5, these feasibility polymorphisms are of the form "max-on-a-subset". Clearly, each component function of a fractional polymorphism has to be a feasibility polymorphism, by Observation 2.4.16. Therefore, the result follows from Theorem 4.5.7. □

Theorem 4.5.9. *For all $d \geq 3$, $\langle \mathbf{G}_{d,1}^{\max} \rangle \subsetneq \langle \mathbf{G}_{d,2}^{\max} \rangle = \mathbf{G}_d^{\max}$. Moreover, $\langle \mathbf{G}_{2,1}^{\max} \rangle \subsetneq \langle \mathbf{G}_{2,2}^{\max} \rangle \subsetneq \langle \mathbf{G}_{2,3}^{\max} \rangle = \mathbf{G}_2^{\max}$.*

Proof. The separation results were obtained in Propositions 4.5.1 and 4.5.2, by showing that the valued constraint languages involved have different feasibility polymorphisms.

For all $d' \geq 2$, $m \geq 1$ and $c \in \mathbb{R}_+$, $\mathbf{G}_{d',m}^{\max}$ is closed under scaling by c. Therefore, using Theorem 2.4.19, the collapses follow from Proposition 4.5.4 and Theorem 4.5.8. □

We now present a non-algebraic proof of the collapse results from Theorem 4.5.9. In fact, we prove a slightly stronger result: for all $d \geq 3$, $\mathbf{G}_d^{\max} = \langle \mathbf{R}_{d,2}^{\max} \cup \mathbf{F}_{d,1}^{\max} \rangle$, and $\mathbf{G}_2^{\max} = \langle \mathbf{R}_{2,3}^{\max} \cup \mathbf{F}_{2,1}^{\max} \rangle$. However, the following proof requires that cost functions take integer values.

Proof. (alternative proof of Theorem 4.5.9)

Let ϕ be an m-ary general max-closed cost function, and write $x_1, \ldots, x_m$ for the variables. Let $y_1, \ldots, y_K$ be variables (with d values, say $1, \ldots, d$), where $K =$

$\max\{\phi(\mathbf{x})|\phi(\mathbf{x}) < \infty\}$ is the biggest finite cost in the range of ϕ. Intuitively, a cost of k for a tuple will be encoded by $y_1, \ldots, y_k$ assigned the value 1 (and the other variables assigned the value 0).

We first encode infinite costs. Let $\phi_R = \mathsf{Feas}(\phi)$ be the relation containing tuples u such that $\phi(u) < \infty$. It turns out that this relation is max-closed. Indeed, for all $u, v \in \phi_R$ we have $\phi(u), \phi(v) < \infty$ by definition of ϕ_R. Since ϕ is max-closed we have $2\phi(\textsc{Max}(u, v)) \leq \phi(u) + \phi(v) < \infty$, so $\phi(\textsc{Max}(u, v)) < \infty$ and thus, $\textsc{Max}(u, v) \in \phi_R$. So ϕ_R is max-closed, that is, $\phi_R \in \mathbf{R}_d^{\max}$.

We now encode finite costs. For an m-tuple t with $\phi(t) < \infty$, write k_t for $\phi(t)$. We let ψ_t be the anti-Horn formula $\bigwedge_{j=1}^{k_t}((\bigvee_{i=1}^{m} x_i > t[i]) \vee y_j \leq 1)$. Observe that this formula reads $\mathbf{x} \leq t \rightarrow y_1 \leq 1 \wedge \cdots \wedge y_{k_t} \leq 1$.

Finally, we define the anti-Horn formula ψ to be $\psi_R \wedge \bigwedge_{t \in D^m} \psi_t$, where ψ_R is an anti-Horn formula equivalent to ϕ_R. By Proposition 4.3.10, this formula can be expressed over $\mathbf{R}_{d,2}^{\max}$.

The formula ψ encodes the cost of every tuple as a number of y_j's assigned the value 1. We thus add, to every variable y_j, $j = 1, \ldots, K$, the cost function μ defined by $\mu(1) = 1$ and $\mu(2) = \cdots = \mu(d) = 0$. Clearly, this function is max-closed and therefore in $\mathbf{F}_{d,1}^{\max}$.

We now show that the gadget $\langle \psi, \langle x_1, \ldots, x_m \rangle \rangle$ expresses ϕ. Let t be an m-ary tuple. Assume first that $k_t = \phi(t)$ is finite. Then ψ contains the subformula

$$\bigwedge_{j=1}^{k_t}((\bigvee_{i=1}^{m} x_i > t[i]) \vee y_j \leq 1).$$

Since obviously $t[i] > t[i]$ holds for no i, every assignment which satisfies ψ sets variables $y_1, \ldots, y_{k_t}$ to 1 and thus, has a cost of at least k_t. Now let s_t be the assignment which is equal to t over $x_1, \ldots, x_m$ and which assigns 1 to $y_1, \ldots, y_{k_t}$ and 2 to $y_{k+1}, \ldots, y_K$. We show that s_t satisfies ψ, which gives an assignment of cost at most k_t.

First let $\psi_{t'} \in \psi$, and recall that $\psi_{t'}$ reads $\mathbf{x} \leq t' \rightarrow y_1 \leq 1 \wedge \cdots \wedge y_{\phi(t')} \leq 1$. If $t \leq t'$, then since ϕ is max-closed and both costs are finite (by definition of $\psi_{t'}$), we have $\phi(t) \geq \phi(t')$ by Proposition 4.4.4. It follows that $\{y_1, \ldots, y_{\phi(t')}\} \subseteq \{y_1, \ldots, y_{k_t}\}$, so s_t assigns 1 to $y_1, \ldots, y_{\phi(t')}$ and thus satisfies $\psi_{t'}$. Otherwise, if $t \not\leq t'$, then $t[i] > t'[i]$ for some i and thus s_t satisfies $\psi_{t'}$ (it does not satisfy its premises). Finally, s_t satisfies $\psi_{t'}$ for all t'.

Now s_t satisfies ψ_R by definition of ϕ_R, since s_t equals t over $x_1, \ldots, x_m$ and $\phi(t) < \infty$ by our assumption. We finally have that for all t with finite cost under ϕ, s_t satisfies ψ and thus, the projection of ψ assigns a cost of at most k_t to t. Since the cost is at least k_t as shown above, we are done.

Now if t has infinite cost under ϕ, then by definition of ϕ_R we have that t does not satisfy ψ and thus, has infinite cost. Therefore, for all $d \geq 3$, $\mathbf{G}_d^{\max} = \langle \mathbf{R}_{d,2}^{\max} \cup \mathbf{F}_{d,1}^{\max} \rangle$,

In the case of a Boolean domain, ψ is a relation over the Boolean domain, and therefore ψ can be expressed, by Proposition 4.3.10, over $\mathbf{R}_{2,3}^{\max}$. Therefore, $\mathbf{G}_2^{\max} = \langle \mathbf{R}_{2,3}^{\max} \cup \mathbf{F}_{2,1}^{\max} \rangle$. □

Figure 4.8 summarises the results from this section.

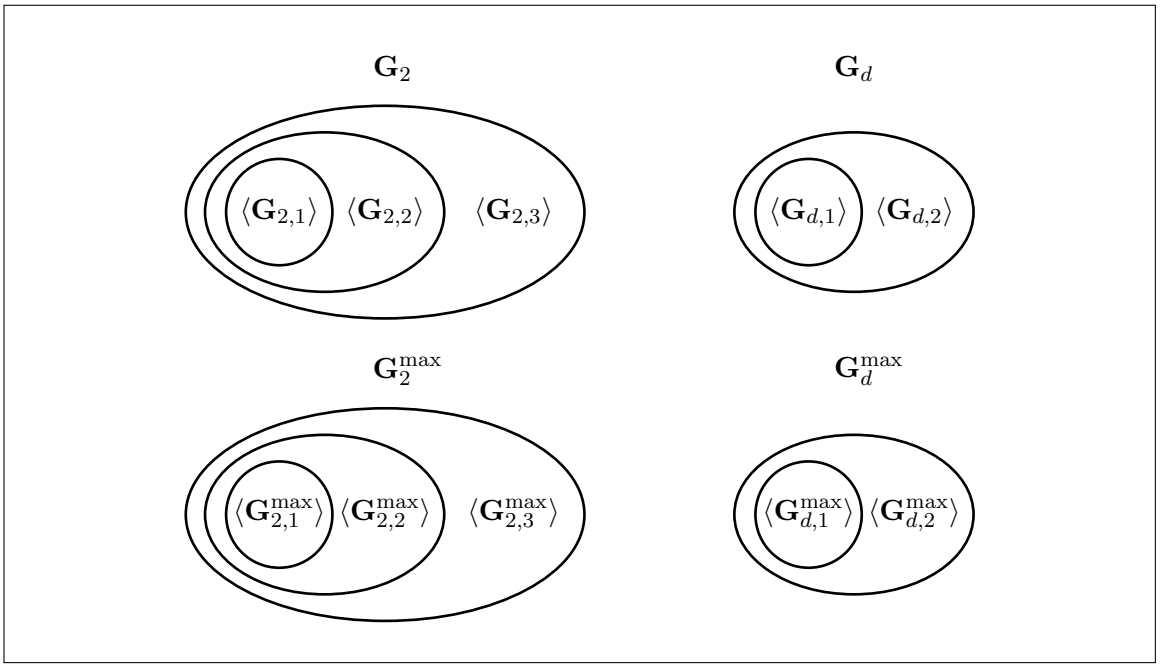

Figure 4.8: Summary of results from Section 4.5, for all $d \geq 3$.

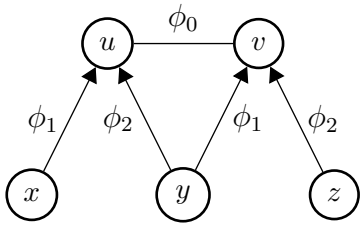

Figure 4.9: $\mathcal{P}_1$, an instance of $\mathsf{VCSP}(\mathbf{G}_{3,2}^{\max})$ expressing ϕ (Example 4.5.10).

Example 4.5.10. Consider the ternary finite-valued max-closed cost function ϕ over $D = \{0, 1, 2\}$ which is defined by

$$\phi(t) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } t = \langle 0, 0, 0 \rangle, \\ 0 & \text{otherwise.} \end{cases}$$

By Proposition 4.4.6, $\phi \notin \langle \mathbf{F}_{3,2}^{\max} \rangle$. In other words, ϕ is not expressible using only finite-valued max-closed cost functions of arity at most 2. However, by Theorem 4.5.9, $\phi \in \langle \mathbf{G}_{3,2}^{\max} \rangle$. We now show how ϕ can be expressed using general max-closed cost functions of arity at most 2.

Let ϕ_0 be the binary finite-valued max-closed cost function defined as follows:

$$\phi_0(t) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } t = \langle 0, 0\rangle, \\ 0 & \text{otherwise.} \end{cases}$$

Next, define two binary crisp[2] max-closed cost functions

$$\phi_1(t) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } t = \langle 0, 1\rangle, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\phi_2(t) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } t = \langle 0, 2\rangle, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\mathcal{P}_1 = \langle V, D, \mathcal{C}\rangle$ where $V = \{x, y, z, u, v\}$ and

$$\mathcal{C} = \{\langle\langle x, u\rangle, \phi_1\rangle, \langle\langle y, u\rangle, \phi_2\rangle, \langle\langle y, v\rangle, \phi_1\rangle, \langle\langle z, v\rangle, \phi_2\rangle, \langle\langle u, v\rangle, \phi_0\rangle\}.$$

We claim that $\langle\mathcal{P}_1, \langle x, y, z\rangle\rangle$ is a gadget for expressing ϕ over $\mathbf{G}^{\max}_{3,2}$. (See Figure 4.9.) If any of x, y, z is non-zero, then at least one of the variables u, v can be assigned a non-zero value and the cost of such an assignment is 0. Conversely, if x, y and z are all assigned zero, then the minimum-cost assignment must also assign zero to both u and v, and hence has cost 1.

We now show another gadget for expressing ϕ, using only crisp max-closed cost functions of arity at most 2 and finite-valued max-closed cost functions of arity at most 1.

Let μ be the unary finite-valued max-closed cost function defined by

$$\mu(x) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\mathcal{P}_2 = \langle V', D, \mathcal{C}'\rangle$ where $V' = \{x, y, z, u, v, w\}$ and

$$\mathcal{C} = \{\langle\langle x, u\rangle, \phi_1\rangle, \langle\langle y, u\rangle, \phi_2\rangle, \langle\langle y, v\rangle, \phi_1\rangle, \langle\langle z, v\rangle, \phi_2\rangle, \langle\langle u, w\rangle, \phi_1\rangle, \langle\langle v, w\rangle, \phi_2\rangle, \langle w, \mu\rangle\}.$$

See Figure 4.10. Similarly to the argument above, $\langle\mathcal{P}_2, \langle x, y, z\rangle\rangle$ is a gadget for expressing ϕ. This can be verified by examining the microstructure of $\mathcal{P}_2$ (see Figure 4.11): circles represent particular assignments to particular variables, as indicated, and edges are weighted by the cost of the corresponding pair of assignments. Thin edges indicate zero weight, bold edges indicate infinite weight, and assigning 0 to variable w gives cost 1.

[2]Note that a "finite variant" of ϕ_1, defined as $\phi_1(\langle 0, 1\rangle) = K$ for some finite $K < \infty$ and $\phi_1(\langle ., .\rangle) = 0$ otherwise, is not max-closed. The infinite cost is necessary.

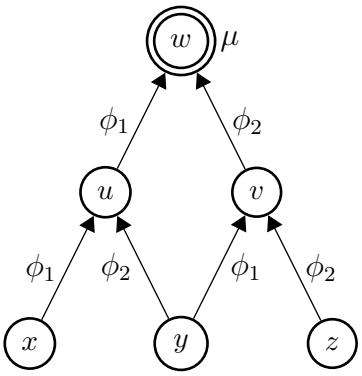

Figure 4.10: $\mathcal{P}_2$, an instance of $\mathsf{VCSP}(\mathbf{R}_{3,2}^{\max} \cup \mathbf{F}_{3,1}^{\max})$ expressing ϕ (Example 4.5.10).

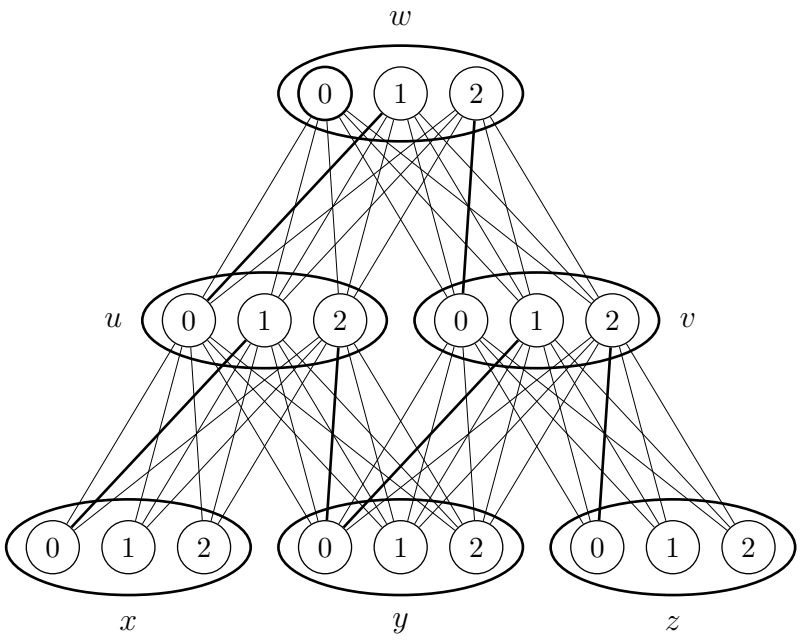

Figure 4.11: Microstructure of the instance $\mathcal{P}_2$ (Example 4.5.10).

Example 4.5.11. Recall Example 2.3.9 which showed how to express $\phi = (\#0)^2$ using cost functions ϕ_0, ϕ_1, ϕ_2 and μ. All three binary crisp cost functions ϕ_0, ϕ_1, and ϕ_2 are max-closed. Moreover, the unary finite-valued cost function μ is max-closed as well.

4.6 Characterisation of $\mathsf{Mul}(\mathbf{F}_d^{\max})$ and $\mathsf{fPol}(\mathbf{F}_d^{\max})$

In section 4.4, we have seen that showing $\mathsf{fPol}(\mathbf{F}_{d,m+1}^{\max}) \not\subseteq \mathsf{fPol}(\mathbf{F}_{d,m}^{\max})$ implies an infinite hierarchy of cost functions with ever-increasing expressive power. In section 4.5, we have characterised the fractional clone of general max-closed cost functions. In this section, we characterise the multi-clone and fractional clone of finite-valued max-closed cost functions.

Notation 4.6.1. Recall from Notation 4.4.3 that we say that an m-tuple u *dominates* an m-tuple v, denoted $u \geq v$, if $u[i] \geq v[i]$ for all $1 \leq i \leq m$. We say that u *strictly dominates* v if $u[i] > v[i]$ for all $1 \leq i \leq m$. If $s \geq t$ ($s > t$ respectively), we also write $t \leq s$ ($t < s$ respectively).

Notation 4.6.2. For a graph $G = (V, E)$ and a set of vertices $V' \subseteq V$, we define the set of neighbours of V' as $N(V') = \{v \in V | (\exists v' \in V)\ [(v, v') \in E)]\}$. We say a graph $G = (V, E)$ is *bipartite* if $V = V_0 \dot\cup V_1$ and $E \subseteq V_0 \times V_1$. For a bipartite graph $G = (V_0, V_1, E)$, a *matching* is a set of edges $E' \subseteq E$ such that no two edges from E' share a vertex, and the size of such a matching is $|E'|$. A *perfect matching* is a matching of size $|V_0|$.

Theorem 4.6.3 (Hall (1935))**.** *A bipartite graph $G = (V_0, V_1, E)$ has a perfect matching if, and only if, $(\forall V' \subseteq V_0)\ [|N(V')| \geq |V'|]$.*

We denote by D a fixed finite totally-ordered domain set with $|D| = d$.

We now characterise the multimorphisms of $\mathbf{F}_d^{\max}$, the valued constraint language containing all finite-valued max-closed cost functions.

Theorem 4.6.4. *Let $\mathcal{F} = \langle f_1, \ldots, f_k \rangle$, where each $f_i : D^k \to D$, and let $S_i = \{j \mid f_i(u_1, \ldots, u_k) \geq u_j\}$, $1 \leq i \leq k$. Then the following are equivalent:*

1. *$\mathcal{F} \in \mathsf{Mul}(\mathbf{F}_d^{\max})$.*
2. *For every fixed collection of k tuples $u_1, \ldots, u_k \in D^m$,*
$$(\forall I \subseteq \{1, \ldots, k\})\ [|\bigcup_{i \in I} S_i| \geq |I|].$$
3. *For every fixed collection of k tuples $u_1, \ldots, u_k \in D^m$, there exists a bijective mapping $\varphi : \{1, \ldots, k\} \to \{1, \ldots, k\}$ such that*
$$(\forall 1 \leq i \leq k)\ [f_i(u_1, \ldots, u_k) \geq u_{\varphi(i)}].$$

Proof. First we show (1) ⇒ (2).

Let $I \subseteq \{1, \ldots, k\}$ and assume for contradiction that $|\cup_{i \in I} S_i| < |I|$. Define a cost function ϕ on D^m as

$$\phi(x) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x \leq f_i(u_1, \ldots, u_k) \text{ for some } i \in I, \\ 0 & \text{otherwise.} \end{cases}$$

We show that ϕ is max-closed. Let u and v be two m-tuples. If $\phi(u) = \phi(v) = 1$, then the multimorphism inequality, as given in Definition 2.4.11, is satisfied easily as ϕ takes only costs 0 and 1. If either $\phi(u) = 0$ or $\phi(v) = 0$ (or both), then $\phi(\textsc{Max}(u, v)) = 0$ from the definition of ϕ, and the multimorphism inequality is satisfied again. Hence ϕ is max-closed.

The following holds from the definition of ϕ:

$$\sum_{i=1}^{k} \phi(u_i) \;=\; |\bigcup_{i \in I} S_i| \;<\; |I| \;\leq\; \sum_{i}^{k} \phi(f_i(u_1, \ldots, u_k)).$$

Therefore, $\mathcal{F}$ is not a multimorphism of ϕ which is a contradiction as ϕ is a finite-valued max-closed cost function.

Now we show (2) ⇒ (3).

Consider a bipartite graph $G = (V_0, V_1, E)$, where

$$\begin{aligned} V_0 =& \{0\} \times \{f_1(u_1, \ldots, u_k), \ldots, f_k(u_1, \ldots, u_k)\}, \\ V_1 =& \{1\} \times \{u_1, \ldots, u_k\}, \end{aligned}$$

and

$$(\langle 0, f_i(u_1, \ldots, u_k)\rangle, \langle 1, u_j\rangle) \in E \;\Leftrightarrow\; (j \in S_i).$$

(Since $\{u_1, \ldots, u_k\}$ and $\{f_1(u_1, \ldots, u_k), \ldots, f_k(u_1, \ldots, u_k)\}$ are not necessarily disjoint, we have to distinguish between the same tuples in these two sets.) We have that for any $V' \subseteq V_0$, $N(V') = \cup_{\{i \mid \langle 0, f_i(u_1, \ldots, u_k)\rangle \in V'\}} S_i$. Therefore, (2) is equivalent to Hall's Condition and by Hall's Theorem 4.6.3, G has a perfect matching. This matching clearly defines the wanted bijective mapping φ.

Finally, we show (3) ⇒ (1).

This is clear as finite-valued max-closed cost functions are finitely antitone by Proposition 4.4.4, and therefore the multimorphism inequality, as given in Definition 2.4.11, is satisfied. □

In other words, a function $\mathcal{F} : D^k \to D^k$ is a multimorphism of $\mathbf{F}_d^{\max}$ if, and only if, for every collection of source k tuples, there is a bijective mapping which maps the k source tuples to the k target tuples in a dominance-preserving way. Note that the same proof characterises the multimorphisms of $\mathbf{F}_{d,m}^{\max}$ for every $m \geq 1$.

Corollary 4.6.5. *$\mathcal{F} \in \mathsf{Mul}(\mathbf{F}_{d,m}^{\max})$ if, and only if, for every collection of k tuples $u_1, \ldots, u_k \in D^m$ there is a bijective mapping $\varphi : \{1, \ldots, k\} \to \{1, \ldots, k\}$ such that $(\forall 1 \leq i \leq k)\ [f_i(u_1, \ldots, u_k) \geq u_{\varphi(i)}]$.*

Proof. Notice that ϕ from the proof of Theorem 4.6.4 is of arity m. Therefore, $\phi \in \mathbf{F}^{\max}_{d,m}$ and the same proof works. □

The next theorem shows that for a given $\mathcal{F} \in \mathsf{Mul}(\mathbf{F}^{\max}_{d})$, there is a uniform bijective mapping from Theorem 4.6.4 (3); that is, a bijective mapping that works for every possible collection of tuples.

Theorem 4.6.6. *For a given $\mathcal{F} = \langle f_1, \ldots, f_k \rangle \in \mathsf{Mul}(\mathbf{F}^{\max}_{d})$, where each $f_i : D^k \to D$, $1 \leq i \leq k$, there is a bijective mapping $\varphi : \{1, \ldots, k\} \to \{1, \ldots, k\}$ such that for every collection of k tuples $u_1, \ldots, u_k \in D^m$, $(\forall 1 \leq i \leq k)$ $[f_i(u_1, \ldots, u_k) \geq u_{\varphi(i)}]$.*

Proof. By Theorem 4.6.4, for every collection of k tuples $u_1, \ldots, u_k \in D^m$, there is a bijective mapping with the required properties. Assume for contradiction that there is no single mapping which preserves dominance for every collection of k m-tuples.

Define a finite-valued max-closed cost function of arity md^{mk} as

$$\phi(x) \stackrel{\text{def}}{=} \psi(x_1, \ldots, x_m) + \psi(x_{m+1}, \ldots, x_{2m}) + \ldots + \psi(x_{(d^{mk}-1)m+1}, \ldots, x_{d^{mk}m})$$

for some $\psi \in \mathbf{F}^{\max}_{d,m}$. Let u and v be two tuples of arity md^{mk}. Since ψ is max-closed, if $u \leq v$, then $\phi(u) \geq \phi(v)$. Hence ϕ is a max-closed cost function. Therefore, $\mathcal{F}$ is a multimorphism of ϕ. By Theorem 4.6.4, there is a bijective dominance-preserving mapping for a collection of k (md^{mk})-tuples which correspond to every possible collection of k m-tuples. □

In a similar way, we can characterise the fractional polymorphisms of $\mathbf{F}^{\max}_{d}$.

Theorem 4.6.7. *Let $\mathcal{F} = \{\langle r_1, f_1 \rangle, \ldots, \langle r_n, f_n \rangle\}$, where each r_i is a positive rational number such that $\sum_{i=1}^{n} r_i = k$ and each $f_i : D^k \to D$. Let $S_i = \{j \mid f_i(u_1, \ldots, u_k) \geq u_j\}$, $1 \leq i \leq k$. Then the following are equivalent:*

1. *$\mathcal{F} \in \mathsf{fPol}(\mathbf{F}^{\max}_{d})$.*
2. *For every fixed collection of k tuples $u_1, \ldots, u_k \in D^m$,*

$$(\forall I \subseteq \{1, \ldots, n\})\ [|\bigcup_{i \in I} S_i| \geq \sum_{i \in I} r_i].$$

Proof. First we show (1) ⇒ (2).

Let $I \subseteq \{1, \ldots, n\}$ and assume for contradiction that $|\cup_{i \in I} S_i| < \sum_{i \in I} r_i$. Define a cost function ϕ on D^m as

$$\phi(x) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x \leq f_i(u_1, \ldots, u_k) \text{ for some } i \in I, \\ 0 & \text{otherwise.} \end{cases}$$

We show that ϕ is max-closed. Let u and v be two m-tuples. If $\phi(u) = \phi(v) = 1$, then the multimorphism inequality, as given in Definition 2.4.11, is satisfied. If either

$\phi(u) = 0$ or $\phi(v) = 0$ (or both), then $\phi(\text{MAX}(u, v)) = 0$ from the definition of ϕ, and the multimorphism inequality is satisfied again. Hence ϕ is max-closed.

The following holds from the definition of ϕ:

$$\sum_{i=1}^{k} \phi(u_i) \;=\; \bigcup_{i \in I} S_i \;<\; \sum_{i \in I} r_i \;\leq\; \sum_{i=1}^{k} r_i \phi(f_i(u_1, \ldots, u_k)).$$

Therefore, $\mathcal{F}$ is not a fractional polymorphism of ϕ which is a contradiction as ϕ is a finite-valued max-closed cost function.

Now we show (2) $\Rightarrow$ (1). Let $u_1, \ldots, u_k \in D^m$ be a fixed collection of k tuples. We want to show that if (2) holds, then

$$\sum_{i=1}^{k} \phi(u_i) \geq \sum_{i=1}^{n} r_i \phi(f_i(u_1, \ldots, u_k)). \tag{4.2}$$

Since all r_i, $1 \leq i \leq n$, are rational numbers, we have $r_i = p_i/q_i$. Let $q = lcm(q_1, \ldots, q_n)$.[3] Then for every $1 \leq i \leq n$, $r_i = ((p_i q)/q_i)(1/q) = k_i(1/q)$, where k_i is a natural number.

For every $1 \leq i \leq n$, we replace the tuple $f_i(u_1, \ldots, u_k)$, which has weight r_i, with k_i copies of the same tuple where each new tuple has weight $1/q$. Since $\sum_{i=1}^{n} r_i = k$, we have kq new tuples and we denote them $v'_1, \ldots, v'_{kq}$. Clearly,

$$\sum_{i=1}^{n} r_i \phi(f_i(u_1, \ldots, u_k)) = \sum_{i=1}^{kq} (1/q) \phi(v'_i).$$

For every $1 \leq i \leq k$, we replace the tuple u_i, which has (implicit) weight 1, with q copies of the same tuple where each new tuple has weight $1/q$. We denote these new tuples $u'_1, \ldots, u'_{kq}$. Clearly,

$$\sum_{i=1}^{k} \phi(u_i) = \sum_{i=1}^{kq} (1/q) \phi(u'_i).$$

Similarly to the proof of Theorem 4.6.4, consider a bipartite graph $G = (V_0, V_1, E)$, where $V_0 = \{v'_1, \ldots, v'_{kq}\}$, $V_1 = \{u'_1, \ldots, u'_{kq}\}$, and $(v'_{i'}, u'_{j'}) \in E \Leftrightarrow v'_{i'}$ replaced some $f_i(u_1, \ldots, u_k)$, and $u'_{j'}$ replaced some u_j and $j \in S_i$.

Clearly, (2) implies Hall's Condition on G. By Theorem 4.6.3, G has a perfect matching. As edges in G preserve dominance on tuples, and every finite-valued max-closed cost function is antitone by Proposition 4.4.4, Inequality (4.2) is satisfied. Therefore, $\mathcal{F} \in \mathsf{fPol}(\mathbf{F}_d^{\max})$. □

[3] least common multiple

4.7 Summary

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 cost functions, where we showed an infinite hierarchy for max-closed cost functions. 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. We also showed the collapse of general cost functions, by characterising the feasibility polymorphisms and fractional polymorphisms of general max-closed cost functions. This shows that allowing infinite costs in max-closed cost functions increases their expressive power substantially, and sometimes allows more finite-valued functions to be expressed.

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 fractional polymorphism $\{\langle 2, \text{MIN}\rangle\}$. 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 fractional polymorphism $\{\langle 1, \text{MIN}\rangle, \langle 1, \text{MAX}\rangle\}$. Hence submodular valued constraints can be characterised as min-max-closed valued constraints. Understanding the expressive power of max-closed valued constraints is a natural first step towards understanding of the expressive power of submodular valued constraints. And indeed, we will study the expressive power of submodular valued constraints in the next two chapters.

Related work We remark on the relationship between our results and some previous work on the VCSP. Larrosa and Dechter have shown [LD00] that both the so-called *dual* representation [DP89] and the *hidden variable* representation [Dec90], which transform any CSP instance into a binary CSP instance, can be generalised to the VCSP framework. However, these representations involve an exponential blow-up (in the arity of the constraints) of the domain size (that is, the set of possible values for each variable). The notion of expressibility that we are using in this thesis always preserves the domain size. Our results clarify which cost functions can be expressed using a given valued constraint language over the same domain, by introducing additional (hidden) variables and constraints; the number of these that are required is fixed for any given cost function.

The class of relations which can be made max-closed by permuting domains of all variables is called *renamable*, *permutable*, or *switchable* anti-Horn. Testing whether a given VCSP instance consists of renamable max-closed cost functions can be done in

polynomial time for Boolean domains [Lew78], but is NP-complete for non-Boolean domains [GC08].

Open problems It would be interesting to know expressive power of various other classes of cost functions of different arities.

CHAPTER 5

Expressive Power of Submodular Functions

We can only see a short distance ahead,
but we can see plenty there that needs to be done.
Alan Turing (1912–1954)

This chapter is based on the following paper:

[ŽJ09a] S. Živný and P.G. Jeavons, Classes of Submodular Constraints Expressible by Graph Cuts. To appear in *Constraints*, 2009.
Earlier version in *Proceedings of the 14th International Conference on Principles and Practice of Contraint Programming (CP'08)*, volume 5202 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2008.

5.1 Introduction

In this chapter, we study the expressive power of binary submodular functions. Our results present known and new classes of submodular functions which are expressible by binary submodular functions.

Recall that SFM_b is the minimisation problem for locally-defined submodular functions. There is a close relationship between the expressive power of binary submodular functions and solving the SFM_b problem via (s,t)-Min-Cut: showing that a class $\mathcal{C}$ of submodular functions is expressible by binary submodular functions is equivalent to showing that the SFM_b problem for functions from $\mathcal{C}$ can be solved via (s,t)-Min-Cut.

As SFM_b is equivalent to VCSP instances with bounded-arity submodular constraints, our results have important consequences for submodular VCSP instances. We will present our results primarily in the language of pseudo-Boolean optimisation. However, in Chapter 6, we will mention the consequences of our results for constraint satisfaction problems and certain optimisation problems arising in computer vision.

5.2 Results

Recall from Section 2.5 that an instance of the SFM problem can be minimised in polynomial time. The time complexity of the fastest known general algorithm for SFM, and therefore for VCSP instances with submodular constraints, is $O(n^6+n^5L)$, where n is the number of variables and L is the look-up time (needed to evaluate the cost of an assignment to all variables) [Orl09].

As discussed in more detail in Section 5.3, we will only deal with Boolean instances of SFM_b. Therefore, an instance of SFM_b with n variables will be represented as a polynomial in n Boolean variables, of some fixed bounded degree. The problem of expressing Boolean submodular functions by binary submodular functions is then equivalent to expressing Boolean submodular polynomials by binary submodular polynomials.

A general algorithm for SFM can always be used for the more restricted SFM_b, but the special features of this more restricted problem sometimes allow more efficient special-purpose algorithms to be used. (Note that we are focusing on *exact* algorithms which find an optimal solution. See [CCJK05] for approximation algorithms for the Max-CSP, which is a special case of the VCSP, and [FMV07] for approximation algorithms for the SFM.) In particular, it has been shown that certain cases can be solved much more efficiently by reducing to the (s,t)-Min-Cut problem; that is, the problem of finding a minimum cut in a directed graph which includes a given source vertex and excludes a given target vertex. For example, it has been known since 1965 that the minimisation of *quadratic* submodular polynomials is equivalent to finding a minimum cut in a corresponding directed graph [Ham65, BH02, CH]. Hence quadratic submodular polynomials can be minimised in $O(n^3)$ time, where n is the number of variables.

A Boolean polynomial in at most 2 variables has degree at most 2, so any *sum* of binary Boolean polynomials has degree at most 2; in other words, it is quadratic. It follows that an efficient algorithm, based on reduction to (s,t)-Min-Cut, can be used to minimise any class of functions that can be written as a sum of binary submodular polynomials. We will say that a polynomial that can be written in this way, perhaps with additional variables to be minimised over, is *expressible* by binary submodular polynomials (see Section 5.3). The following classes of functions have all been shown to be expressible by binary submodular polynomials in this way:[1]

- polynomials where all terms of degree 2 or more have negative coefficients (also known as *negative-positive* polynomials) [Rhy70];
- cubic submodular polynomials [BM85];
- $\{0,1\}$-valued submodular functions (also known as 2-monotone functions) [CKS01, CCJK05];

[1] In fact, it is known that *all* Boolean polynomials (of arbitrary degree) are expressible by binary polynomials [Ros75, BH02], but the general construction does not preserve submodularity; that is, the resulting binary polynomials are not necessarily submodular.

All these classes of functions have been shown to be expressible by binary submodular polynomials and hence minimisable in cubic time (in the total number of variables). Moreover, several classes of submodular functions over non-Boolean domains have also been shown to be expressible by binary submodular functions and hence minimisable in cubic time [BKR96, CCJK04, CCJK05].

Our results are twofold. First, we provide alternative, and often much simpler, proofs of the expressibility results for the above classes of functions. Second, we present a new class of submodular functions of arbitrary arities expressible by binary submodular polynomials, and hence minimisable in cubic time (in the total number of variables).

This chapter is organised as follows. Section 5.3 provides necessary information on submodular functions. In Section 5.4, we show equivalence between the (s,t)-Min-Cut problem and the minimisation problem for quadratic submodular polynomials. In Section 5.5, we present alternative proofs of known expressibility results for so-called negative-positive, $\{0,1\}$-valued and ternary submodular functions. In Section 5.6, we present a new class of submodular functions of arbitrary arities expressible by binary submodular functions.

5.3 Preliminaries

Recall that a cost function of arity n is just a mapping from D^n to $\overline{\mathbb{R}}$ for some fixed finite set D. Cost functions can be added and multiplied by arbitrary positive real values, hence for any given set of cost functions, Γ, we define the convex cone generated by Γ, as follows.

Definition 5.3.1 (Cone)**.** For any set of cost functions Γ, the *cone generated by* Γ, denoted $\mathsf{Cone}(\Gamma)$, is defined by:

$$\mathsf{Cone}(\Gamma) \stackrel{\text{def}}{=} \{\alpha_1\phi_1 + \cdots + \alpha_r\phi_r \mid r \geq 1;\ \phi_1, \ldots, \phi_r \in \Gamma;\ \alpha_1, \ldots, \alpha_r \geq 0\}.$$

Note that Definition 2.3.1 of expressibility can be stated equivalently as follows:

Definition 5.3.2 (Expressibility)**.** A cost function ϕ of arity n is said to be *expressible* by a set of cost functions Γ if $\phi = \min_{y_1,\ldots,y_j} \phi'(x_1, \ldots, x_n, y_1, \ldots, y_j)$, for some $\phi' \in \mathsf{Cone}(\Gamma)$. The variables $y_1, \ldots, y_j$ are called *extra* (or *hidden*) variables, and ϕ' is called a *gadget* for ϕ over Γ.

Remark 5.3.3. Recall from Definition 2.4.1 and Theorem 2.4.19 that we care about expressibility up to additive and multiplicative constants.

Lemma 5.3.4 ([CCJK06])**.** *Let D be a finite lattice-ordered set. A cost function $\phi : D^m \to \overline{\mathbb{R}}$ is submodular if, and only if, the following two conditions are satisfied:*

1. *ϕ satisfies that for all m-tuples u, v with $\phi(u), \phi(v) < \infty$,*

$$\phi(\textsc{Min}(u,v)) + \phi(\textsc{Max}(u,v)) \leq \phi(u) + \phi(v).$$

2. $\textsc{Min}, \textsc{Max} \in \mathsf{FPol}(\{\phi\})$.

The second condition in Lemma 5.3.4 implies that the set of m-tuples on which ϕ is finite is a sublattice of the set of all m-tuples, where the lattice operations are the operations Min and Max. Theorem 49.2 of [Sch03] proves that any real-valued submodular function defined on such a sublattice can be extended to a submodular function defined on the full lattice.[2] Hence, by Lemma 5.3.4, any submodular function ϕ can be expressed as the sum of a finite-valued submodular function ϕ_{fin}, and a submodular relation $\phi_{\mathsf{rel}} = \mathsf{Feas}(\phi)$, that is, $\phi = \phi_{\mathsf{fin}} + \phi_{\mathsf{rel}}$.

It is known that all submodular *relations* are binary decomposable (that is, equal to the sum of their binary projections) [JCC98], and hence expressible using only binary submodular relations. Therefore, when considering which cost functions are expressible by binary submodular cost functions, we can restrict our attention to *finite-valued* cost functions without any loss of generality.

Remark 5.3.5. We discuss more the restriction to finite-valued submodular cost functions. Given a finite lattice-ordered set D, let ϕ be a submodular cost function defined on a sublattice D' of D. The goal is to change the definition of f to the whole of D so that the resulting cost function is submodular, and the minimisation problem is not affected by these changes. In other words, we would like to find a finite-valued submodular $\bar{f}$ such that $f = \bar{f}$ on D', $\min f = \min \bar{f}$, and the minimum is achieved on D'.

Schrijver has shown [Sch03] that for a given ϕ as above, there is an $\alpha \in \mathbb{R}_+$ such that $\bar{f}(u) = f(\bar{u}) + \alpha|\bar{u} \triangle u|$ is a finite-valued submodular cost function, where $\bar{u}$ is the smallest element above u such that $f(\bar{u}) < \infty$, and $\bar{u} \triangle u$ is the symmetric difference between the sets corresponding to $\bar{u}$ and u in the 0-1 representation of D. Clearly, the same holds for every $\alpha' \geq \alpha$. For example, in a VCSP instance with submodular valued constraints over n variables, it is sufficient to choose $\alpha' \geq nM$, where M is the maximum finite cost of all cost functions.

We have shown that when dealing with the expressibility problem for submodular cost functions, we can restrict ourselves to only finite-valued cost functions without any loss of generality. Now we show that we can restrict ourselves to only *Boolean* finite-valued cost functions.

Remark 5.3.6. Note that any variable over a non-Boolean domain $D = \{0, 1, \ldots, d-1\}$ of size d can be encoded by $d-1$ Boolean variables. This process is known as *Booleanisation*. One such encoding is the following: $en(i) = 0^{d-i-1}1^i$. Using this encoding function we can replace each variable with $d-1$ new Boolean variables and impose a (submodular) relation on these new variables which ensures that they only take values in the range of the encoding function en. Note that $en(\max(a, b)) =$

[2] Note that this result is not obvious because simply changing the infinite cost to some big, but finite constant M does not work: for $c_1 < c_2$, $\infty + c_1 \geq \infty + c_2$, but $M + c_1 < M + c_2$. For instance, consider the submodular cost function ϕ defined as follows: $\phi(0,0) = \phi(1,0) = \infty$, $\phi(0,1) = 1$ and $\phi(1,1) = 2$. Changing $\phi(0,0) = \phi(1,0) = M$ for any finite number M would violate the submodularity condition.

$\max(en(a), en(b))$ and $en(\min(a, b)) = \min(en(a), en(b))$, so this encoding preserves submodularity. Therefore, in this chapter, we will focus on problems over Boolean domains, that is, where $D = \{0, 1\}$.

Remark 5.3.7. Because every chain in a lattice-ordered set has to be mapped to a chain, and different chains have to be mapped to different chains, any submodularity-preserving encoding of a non-Boolean variable over a d-element domain by Boolean variables needs at least d variables. However, for practical purposes, certain subclasses of non-Boolean submodular functions which can be encoded by Boolean submodular functions with fewer variables have been studied, as well as approximation algorithms for these problems [RKAT08, KLT09].

Any cost function of arity m can be represented as a table of values of size D^m. Moreover, a finite-valued cost function $\phi : D^m \to \mathbb{R}$ on a Boolean domain $D = \{0, 1\}$ can also be represented as a *polynomial* in m (Boolean) variables with coefficients from $\mathbb{R}$, and the degree of this polynomial is at most m (such functions are sometimes called *pseudo-Boolean functions* [BH02, CH]). Over a Boolean domain we have $x^2 = x$, so the degree of any variable in any term can be restricted to 0 or 1, and this polynomial representation is then unique. Hence, in what follows, we will often refer to a finite-valued cost function on a Boolean domain and its corresponding polynomial interchangeably.

For polynomials over Boolean variables there is a standard way to define *derivatives* of each order (see [BH02, CH]). For example, the second-order derivative of a polynomial p, with respect to the first two indices, denoted $\delta_{12}(\mathbf{x})$, is defined as $p(1, 1, \mathbf{x}) - p(1, 0, \mathbf{x}) - p(0, 1, \mathbf{x}) + p(0, 0, \mathbf{x})$. Derivatives for other pairs of indices are defined analogously. It has been shown in [NWF78] that a polynomial $p(x_1, \ldots, x_n)$ over Boolean variables $x_1, \ldots, x_n$ represents a submodular cost function if, and only if, its second-order derivatives $\delta_{ij}(\mathbf{x})$ are non-positive for all $1 \leq i < j \leq n$ and all $\mathbf{x} \in D^{n-2}$. An immediate corollary is that a quadratic polynomial represents a submodular cost function if, and only if, the coefficients of all quadratic terms are non-positive.

Remark 5.3.8. Note that a cost function is called *supermodular* if all its second-order derivatives are non-negative. Clearly, f is submodular if, and only if, $-f$ is supermodular, so it is straightforward to translate results about supermodular functions, such as those given in [CCJK05] and [PY05], into similar results for submodular functions, and we will use this observation several times below. Cost functions which are both submodular and supermodular (in other words, all second-order derivatives are equal to zero) are called *modular*, and polynomials corresponding to modular cost functions are linear [BH02, CH].

Example 5.3.9. For any set of indices $I = \{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\}$ we can define a cost function ϕ_I in n variables as follows:

$$\phi_I(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \begin{cases} -1 & \text{if } (\forall i \in I)(x_i = 1), \\ 0 & \text{otherwise.} \end{cases}$$

The polynomial representation of ϕ_I is $p(x_1, \ldots, x_n) = -x_{i_1} \ldots x_{i_m}$, which is a polynomial of degree m. By checking the second-order derivatives of p, it follows that ϕ_I is submodular.

However, the function ϕ_I is also expressible by *binary* polynomials, using a single extra variable, y, as follows:

$$\phi_I(x_1, \ldots, x_n) = \min_{y \in \{0,1\}} y(m - 1 - \sum_{i \in I} x_i).$$

We remark that this is a special case of the expressibility result for negative-positive polynomials first obtained in [Rhy70].

Definition 5.3.10. We denote by $\Gamma_{\mathsf{sub},k}$ the set of all finite-valued submodular cost functions of arity at most k on a Boolean domain D, and we set $\Gamma_{\mathsf{sub}} \stackrel{\text{def}}{=} \bigcup_k \Gamma_{\mathsf{sub},k}$.

5.4 Reduction to (s,t)-Min-Cut

In this section, we show equivalence between the minimisation problem for quadratic submodular polynomials and the (s,t)-Min-Cut problem.

Theorem 5.4.1 ([Ham65])**.** (s,t)-Min-Cut *and the minimisation problem of polynomials over* $\Gamma_{\mathsf{sub},2}$ *are polynomial-time equivalent.*

Proof. First we show that any instance $\langle G = \langle V, E\rangle, w, s, t\rangle$, where $w : E \to \mathbb{R}_+$ and $s, t \in V$, of (s,t)-Min-Cut is reducible to the minimisation problem for quadratic submodular polynomials:

- every vertex from V is represented by a single Boolean variable;
- by adding a linear term Ms, for large M, we impose a unary constraint on s to take the value 0;
- similarly, by adding a linear term $M(1-t)$, we impose a unary constraint on t to take the value 1;
- for every edge $\langle u, v\rangle \in E$, we add a submodular quadratic term $av - auv$, where $a = w(\langle u, v\rangle)$ is the weight of the edge $\langle u, v\rangle$ in G (note that $av - auv$ returns a if, and only if, $u = 0$ and $v = 1$, and 0 otherwise).

There is equivalence between (s,t)-cuts in G, that is, subsets of vertices including s but excluding t, and assignments of zeros and ones to variables in the corresponding polynomial which set s to 0 and t to 1. (Value 0 corresponds to vertices in the cut.)

On the other hand, we show now that any submodular quadratic polynomial can be minimised by reducing to (s,t)-Min-Cut.

Let p be an arbitrary submodular quadratic polynomial, that is,

$$p(x_1, \ldots, x_n) = a_0 + \sum_{i=1}^{n} a_i x_i + \sum_{1 \le i < j \le n} a_{ij} x_i x_j,$$

where $a_{ij} \leq 0$ for all $1 \leq i < j \leq n$. Then,

$$p = a'_0 + \sum_{i \in P} a'_i x_i + \sum_{j \in N} a'_j (1 - x_j) + \sum_{1 \leq i < j \leq n} a'_{ij} (1 - x_i) x_j,$$

where $P \cap N = \emptyset$, $P \cup N = \{1, 2, \ldots, n\}$, $a'_{ij} = -a_{ji}$, and $a'_i, a'_j, a'_{ij} \geq 0$. (This is known as a *posiform* [BH02, CH].)

Now p can be easily minimised by reducing to (s,t)-MIN-CUT:

1. vertices of the graph are $x_1, \ldots, x_n$ and two extra vertices s and t;
2. there is an edge going from x_i to x_j with weight a'_{ij};
3. for every $i \in P$, there is an edge going from s to x_i with weight a'_i;
4. for every $j \in N$, there is an edge going from x_j to t with weight a'_j.

Again, there is equivalence between (s,t)-cuts in the constructed graph and assignments (which set s to 0 and t to 1) of zeros and ones to the posiform representation of p. □

Corollary 5.4.2. *A quadratic submodular polynomial in n Boolean variables can be minimised in $O(n^3)$ time.*

Proof. By Theorem 5.4.1, and using some standard cubic-time algorithm for (s,t)-MIN-CUT [GT88]. □

Example 5.4.3. Consider the following quadratic submodular polynomial:

$$\begin{aligned} p = {} & 8 + 12x_1 + 7x_2 + 11x_3 - 5x_4 - 7x_5 \\ & - x_1x_2 - 7x_1x_4 - 3x_2x_3 - 4x_3x_4 - 5x_3x_5 - x_4x_5. \end{aligned}$$

We can rewrite p as in the proof of Theorem 5.4.1 as follows:

$$\begin{aligned} p = {} & 8 + 12x_1 + 7x_2 + 11x_3 - 5x_4 - 7x_5 \\ & + (1 - x_1)x_2 - x_2 + 7(1 - x_1)x_4 - 7x_4 + 3(1 - x_2)x_3 - 3x_3 \\ & + 4(1 - x_3)x_4 - 4x_4 + 5(1 - x_3)x_5 - 5x_5 + (1 - x_4)x_5 - x_5 \\ = {} & -21 + 12x_1 + 6x_2 + 8x_3 + 16(1 - x_4) + 13(1 - x_5) \\ & + (1 - x_1)x_2 + 7(1 - x_1)x_4 + 3(1 - x_2)x_3 + 4(1 - x_3)x_4 + 5(1 - x_3)x_5 + (1 - x_4)x_5. \end{aligned}$$

We can now build a graph G with 5 vertices corresponding to variables x_1 through x_5 and two extra vertices s and t and add edges accordingly (see Figure 5.1).

For every assignment v of values 0 and 1 to variables $x_1, \ldots, x_5$, $p(v(x_1), \ldots, v(x_5))$ is equal to the size of the (s,t)-cut in G given by v minus 21 (for the constant term in the posiform representation of p). The minimum cut in G, with value 16, is the set $\{s, x_1, x_2, x_3\}$. Therefore, the assignment $x_1 = x_2 = x_3 = 0$ and $x_4 = x_5 = 1$ minimises the polynomial p with total value $16 - 21 = -5$.

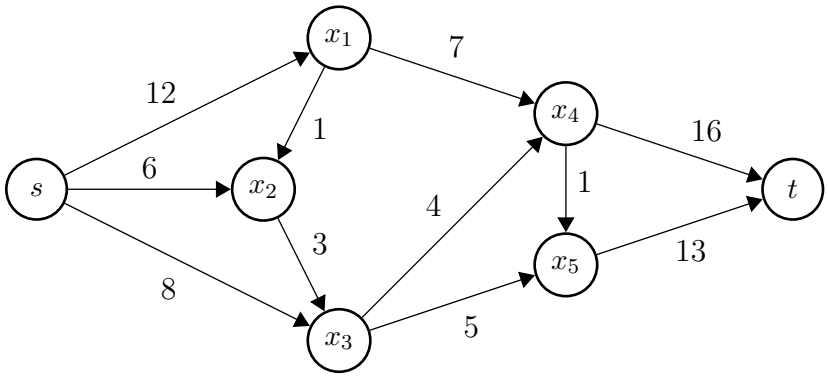

Figure 5.1: Graph G corresponding to polynomial p (Example 5.4.3).

5.5 Known classes of expressible functions

In this section, we present new and simpler proofs for some known expressibility results. We will focus on three classes of cost function on a Boolean domain: submodular cost functions whose corresponding polynomials have negative coefficients for all terms of degree ≥ 2; $\{0,1\}$-valued submodular cost functions; and ternary submodular cost functions. We show that cost functions from these three classes are expressible over $\Gamma_{\mathsf{sub},2}$.

Definition 5.5.1. Define $\Gamma_{\mathsf{neg},k}$ to be the set of all cost functions over a Boolean domain, of arity at most k, whose corresponding polynomials have negative coefficients for all terms of degree greater than or equal to 2. Set $\Gamma_{\mathsf{neg}} \stackrel{\text{def}}{=} \bigcup_k \Gamma_{\mathsf{neg},k}$.

It is well known that these cost functions, sometimes called *negative-positive*, are submodular [BH02]. The minimisation of cost functions chosen from Γ_{neg} using min-cuts was first studied in [Rhy70], but we give a simplified proof.

Theorem 5.5.2 ([Rhy70])**.** $\Gamma_{\mathsf{neg}} \subseteq \langle \Gamma_{\mathsf{sub},2} \rangle$.

Proof. We use the gadget we have already seen in Example 5.3.9: Given any polynomial p representing a cost function from Γ_{neg}, we can replace each term $-ax_{i_1} \dots x_{i_k}$ of p of degree $k \geq 3$, where $a > 0$, by

$$-ax_{i_1} \dots x_{i_k} = \min_{y \in \{0,1\}} \{a(-y + y \sum_{j \in \{1,\dots,k\}} (1 - x_{i_j})\}.$$

In this way, we introduce a new variable for every term of degree ≥ 3. □

Corollary 5.5.3. *A polynomial p in n Boolean variables over Γ_{neg} can be minimised in $O((n+r)^3)$ time, where r is the number of terms of degree 3 or greater in p.*

Next we consider the class of cost functions over a Boolean domain which take only the cost values 0 and 1. (Such constraints can be used to model optimisation problems such as Max-CSP, see [CCJK05].)

Definition 5.5.4. Define $\Gamma_{\{0,1\},k}$ to be the set of all $\{0,1\}$-valued submodular cost functions over a Boolean domain, of arity at most k, and set $\Gamma_{\{0,1\}} \stackrel{\text{def}}{=} \cup_k \Gamma_{\{0,1\},k}$.

The minimisation of submodular cost functions from $\Gamma_{\{0,1\}}$ has been studied in [CKS01], where they have been called *2-monotone* functions. The equivalence of 2-monotone and submodular cost functions and a generalisation of 2-monotone functions to non-Boolean domains[3] has been shown in [CCJK05].

Definition 5.5.5. A cost function ϕ is called *2-monotone* if there exist two sets $A, B \subseteq \{1, \ldots, n\}$ such that $\phi(\mathbf{x}) = 0$ if $A \subseteq \mathbf{x}$ or $\mathbf{x} \subseteq B$ and $\phi(\mathbf{x}) = 1$ otherwise (where $A \subseteq \mathbf{x}$ means $\forall i \in A, x_i = 1$ and $\mathbf{x} \subseteq B$ means $\forall i \notin B, x_i = 0$).

Using the notion of expressive power we are able to give a very simple proof of the following result:

Theorem 5.5.6 ([CKS01])**.** $\Gamma_{\{0,1\}} \subseteq \langle \Gamma_{\mathsf{sub},2} \rangle$.

Proof. Any 2-monotone cost function ϕ can be expressed over $\Gamma_{\mathsf{sub},2}$ using 2 extra variables, y_1, y_2:

$$\phi(\mathbf{x}) = \min_{y_1, y_2 \in \{0,1\}} \{(1 - y_1)y_2 + y_1 \sum_{i \in A} (1 - x_i) + (1 - y_2) \sum_{i \notin B} x_i\}.$$

□

Corollary 5.5.7. *A polynomial p in n Boolean variables over $\Gamma_{\{0,1\}}$ can be minimised in $O((n + r)^3)$ time, where r is the number of 2-monotone cost functions represented in p.*

Finally, we consider the class $\Gamma_{\mathsf{sub},3}$ of ternary submodular cost functions over a Boolean domain. This class has been studied in [BM85], from where we obtain the following useful characterisation of cubic submodular polynomials.

Lemma 5.5.8 ([BM85])**.** *A cubic polynomial $p(x_1, \ldots, x_n)$ over Boolean variables represents a submodular cost function if, and only if, it can be written as*

$$p(x_1, \ldots, x_n) = a_0 + \sum_{\{i\} \in C_1^+} a_i x_i - \sum_{\{i\} \in C_1^-} a_i x_i - \sum_{\{i,j\} \in C_2} a_{ij} x_i x_j + \sum_{\{i,j,k\} \in C_3^+} a_{ijk} x_i x_j x_k - \sum_{\{i,j,k\} \in C_3^-} a_{ijk} x_i x_j x_k,$$

where C_2 denotes the set of quadratic terms, and C_i^+ (C_i^-) denotes the set of terms of degree i with positive (negative) coefficients, for $i = 1, 3$, and

1. *$a_i, a_{ij}, a_{ijk} \geq 0$ ($\{i\} \in C_1^+ \cup C_1^-, \{i,j\} \in C_2, \{i,j,k\} \in C_3^+ \cup C_3^-$), and*

[3] In [CCJK05], 2-monotone cost functions are defined over lattice-ordered sets and called *generalised 2-monotone* cost functions. In [KL08], these are called just *2-monotone* cost functions.

2. $\forall\{i,j\} \in C_2$, $a_{ij} + \sum_{k|\{i,j,k\}\in C_3^+} a_{ijk} \leq 0$.

This characterisation, together with the notion of expressive power, allows us to obtain a very simple proof of the following result:

Theorem 5.5.9 ([BM85])**.** $\Gamma_{\mathsf{sub},3} \subseteq \langle \Gamma_{\mathsf{sub},2} \rangle$.

Proof. Let p be a polynomial representing an arbitrary cost function in $\Gamma_{\mathsf{sub},3}$. By submodularity, all quadratic terms in p are non-positive. We already know how to express a negative cubic term using a gadget over $\Gamma_{\mathsf{sub},2}$ (Theorem 5.5.2). To express a positive cubic term, consider the following identity:

$$x_i x_j x_k - x_i x_j - x_i x_k - x_j x_k = \min_{y\in\{0,1\}} \{(1 - x_i - x_j - x_k)y\}.$$

Hence, we can replace a positive cubic term $a_{ijk}x_i x_j x_k$ in p with

$$\min_{y\in\{0,1\}} \{a_{ijk}(1 - x_i - x_j - x_k)y + a_{ijk}(x_i x_j + x_i x_k + x_j x_k)\}.$$

It remains to check that all quadratic coefficients of the resulting polynomial are non-positive. However, this is ensured by the second condition from Lemma 5.5.8. □

Corollary 5.5.10. *A cubic submodular polynomial p in n Boolean variables can be minimised in $O((n+r)^3)$, where r is the number of cubic terms in p.*

Remark 5.5.11. We remark that the proof of Theorem 5.5.9 given in [BM85] was obtained using a different approach based on the so-called conflict graphs of a supermodular polynomial (see [BH02, CH]). Such graphs have been shown to be bipartite, and therefore the problem of finding a maximum weight stable set can be reduced to a flow problem. However, the resulting time complexity is the same.

5.6 New classes of expressible functions

In this section, we present new classes of submodular cost functions which can be expressed by binary submodular cost functions. First, we derive a necessary condition for a 4-ary cost function over a Boolean domain to be submodular. We also present some sufficient conditions, which give rise to new classes of submodular cost functions which can be expressed over $\Gamma_{\mathsf{sub},2}$, and hence minimised efficiently. We prove the sufficient conditions first for 4-ary submodular cost functions and then generalise them to k-ary submodular cost functions for every $k \geq 4$.

We start with submodular cost functions of arity 4. One might hope to obtain a simple characterisation of 4-ary submodular cost functions over a Boolean domain similar to Lemma 5.5.8. However, it has been shown that testing whether a given quartic Boolean polynomial is submodular is co-NP-complete [Cra89, GS88]. Hence, one is unlikely to find a polynomial-time checkable characterisation, as this would prove that P=NP. However, we obtain the following necessary condition.

Lemma 5.6.1. *If a quartic polynomial $p(x_1, \ldots, x_n)$ over Boolean variables represents a submodular cost function, then it can be written such that, for all $\{i, j\} \in C_2$:*

1. $a_{ij} \leq 0$, *and*
2. $a_{ij} + \sum_{k|\{i,j,k\}\in C_3^+} a_{ijk} + \sum_{k,l|\{i,j,k,l\}\in C_4^+} a_{ijkl} + F_{ij} \leq 0$, *where*

$$F_{ij} = \sum_{k|\{i,j,k\}\in C_3^- \wedge \{i,j,k,.\}\in C_4^+} a_{ijk} + \sum_{k,l|\{i,j,k,l\}\in C_4^- \wedge \{i,j,k,.\},\{i,j,l,.\}\in C_4^+} a_{ijkl},$$

C_2 denotes the set of quadratic terms, and C_i^+ (C_i^-) denotes the set of terms of degree i with positive (negative) coefficients, for $i = 3, 4$.

Proof. Let p be a quartic submodular polynomial and let i and j be given, then $\delta_{ij}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n)$, the second-order derivative of p with respect to the ith and jth variable, is equal to

$$\delta_{ij} = a_{ij} + \sum_{k|\{i,j,k\}\in C_3^+} a_{ijk}x_k + \sum_{k,l|\{i,j,k,l\}\in C_4^+} a_{ijkl}x_k x_l - \sum_{k|\{i,j,k\}\in C_3^-} a_{ijk}x_k - \sum_{k,l|\{i,j,k,l\}\in C_4^-} a_{ijkl}x_k x_l.$$

Consider an assignment which sets $x_k = 1$ if $k = i$ or $k = j$, and $x_k = 0$ otherwise. From submodularity, $a_{ij} \leq 0$, which proves the first condition. By setting $x_k = 1$ for all k such that $\{i, j, k\} \in C_3^+$ and $x_k = x_l = 1$ for all k, l such that $\{i, j, k, l\} \in C_4^+$, we get the second condition. We set to 1 all variables which occur in some positive cubic or quartic term. The second condition then says that the sum of all these positive coefficients minus those which are forced, by our setting of variables, to be 1 (F_{ij}), is at most 0. (Note that this also proves Lemma 5.5.8.) □

Next we show a useful example of a 4-ary submodular cost function which can be expressed over the binary submodular cost functions using one extra variable.

Example 5.6.2. Let ϕ be the 4-ary cost function defined as follows: $\phi(\mathbf{x}) = \min\{2k, 5\}$, where k is the number of 0s in $\mathbf{x} \in \{0, 1\}^4$. The corresponding quartic polynomial representing ϕ is

$$p(x_1, x_2, x_3, x_4) = 5 + x_1x_2x_3x_4 - x_1x_2 - x_1x_3 - x_1x_4 - x_2x_3 - x_2x_4 - x_3x_4.$$

By considering second-order derivatives of p, it can be checked that p is submodular. For instance, $\delta_{12}(x_3, x_4)$, the second-order derivative of p with respect to the first two variables, is equal to $x_1x_2x_3x_4 - 1$. Clearly, $\delta_{12}(x_3, x_4) \leq 0$. It can be shown by a simple case analysis that p cannot be expressed as a quadratic polynomial with non-positive quadratic coefficients (from the definition of p, the polynomial would have to be $5 - x_1x_2 - x_1x_3 - x_1x_4 - x_2x_3 - x_2x_4 - x_3x_4$ which is not equal to p on $x_1 = x_2 = x_3 = x_4 = 1$).

However, p can be expressed over $\Gamma_{\mathsf{sub},2}$ using just one extra variable, via the following gadget:

$$p(x_1, x_2, x_3, x_4) = \min_{y \in \{0,1\}} \{5 + (3 - 2x_1 - 2x_2 - 2x_3 - 2x_4)y\}.$$

Definition 5.6.3. Using the same notation as in Lemma 5.6.1, define $\Gamma_{\mathsf{suff},4}$ to be the set of all 4-ary submodular cost functions over a Boolean domain whose corresponding quartic polynomials satisfy, for every $i < j$,

$$a_{ij} + \sum_{k \mid \{i,j,k\} \in C_3^+} a_{ijk} + \sum_{k,l \mid \{i,j,k,l\} \in C_4^+} a_{ijkl} \quad \leq \quad 0. \tag{5.1}$$

Theorem 5.6.4. $\Gamma_{\mathsf{suff},4} \subseteq \langle \Gamma_{\mathsf{sub},2} \rangle$.

Proof. Let $\phi \in \Gamma_{\mathsf{suff},4}$ and let p be the corresponding polynomial which represents ϕ. First, replace all negative cubic and quartic terms using the construction in Theorem 5.5.2. As in the proof of Theorem 5.5.9, replace every positive cubic term $a_{ijk}x_ix_jx_k$ in p with

$$\min_{y \in \{0,1\}} \{a_{ijk}(1 - x_i - x_j - x_k)y + a_{ijk}(x_ix_j + x_ix_k + x_jx_k)\}.$$

Using the same construction as in Example 5.6.2, replace every positive quartic term $a_{ijkl}x_ix_jx_kx_l$ with

$$\min_{y \in \{0,1\}} \{a_{ijkl}(3 - 2x_i - 2x_j - 2x_k - 2x_l)y + a_{ijkl}(x_ix_j + x_ix_k + x_ix_l + x_jx_k + x_jx_l + x_kx_l)\}.$$

It only remains to check that all quadratic coefficients in the resulting polynomial are non-positive. However, this is ensured by the definition of $\Gamma_{\mathsf{suff},4}$ and by the choice of the gadgets. □

Our next example shows that $\Gamma_{\mathsf{suff},4} \subsetneq \Gamma_{\mathsf{sub},4}$.

Example 5.6.5. Define a 4-ary submodular cost function ϕ as follows: $\phi(\mathbf{x}) = \min(3k, 7) + 2y + z$, where k is the number of 0s in $\mathbf{x} \in \{0,1\}^4$; $y = 1$ if, and only if, $\mathbf{x} = \langle 1,1,1,0 \rangle$ (and 0 otherwise), and $z = 1$ if, and only if, $\mathbf{x} = \langle 1,1,0,0 \rangle$ (and 0 otherwise). The corresponding polynomial representing ϕ is

$$p(x_1, x_2, x_3, x_4) = 7 + 2x_1x_2x_3x_4 - 2x_1x_2x_4 - x_1x_3x_4 - x_2x_3x_4 - x_1x_3 - x_1x_4 - x_2x_3 - x_2x_4 - x_3x_4.$$

By considering the second-order derivatives of p, it can easily be checked that ϕ is submodular. However, $\phi \notin \Gamma_{\mathsf{suff},4}$: for $i = 1$ and $j = 2$, the expression in Equation 5.1 on page 93 gives 2. Hence Theorem 5.6.4 does not apply to ϕ.

As in Example 5.6.2, by a simple case analysis (system of equations), it can be shown that ϕ cannot be expressed over $\Gamma_{\mathsf{sub},2}$ without extra variables or with just one

extra variable. However, the following gadget shows that ϕ is in fact expressible over $\Gamma_{\mathsf{sub},2}$ using just two extra variables:

$$p(x_1,x_2,x_3,x_4) = 7 - x_1x_4 - x_2x_4 - x_3x_4 \\ + \min_{y_1,y_2\in\{0,1\}}\{2y_1 + 3y_2 - y_1y_2 - y_1(x_1+x_2+2x_3) - y_2(x_1+x_2+2x_4)\}.$$

We now generalise the expressibility result of submodular cost functions from $\Gamma_{\mathsf{suff},k}$ to subclasses of submodular cost functions of arbitrary arities.

Definition 5.6.6. We define $\Gamma_{\mathsf{suff},k}$ to be the set of all k-ary submodular cost functions over a Boolean domain whose corresponding polynomials satisfy, for every $1 \le i < j \le k$,

$$a_{ij} + \sum_{s=1}^{k-2} \sum_{\{i,j,i_1,\ldots,i_s\}\in C^+_{s+2}} a_{i,j,i_1,\ldots,i_s} \quad \le \quad 0,$$

where C_i^+ denotes the set of terms of degree i with positive coefficients.

In other words, for any $1 \le i < j \le k$, the sum of a_{ij} and all positive coefficients of cubic and higher-degree terms which include x_i and x_j is non-positive.

Theorem 5.6.7. *For every $k \ge 4$, $\Gamma_{\mathsf{suff},k} \subseteq \langle\Gamma_{\mathsf{sub},2}\rangle$.*

Proof. First we show how to uniformly generate gadgets over $\Gamma_{\mathsf{sub},2}$ for polynomials of the following type:

$$p_k(x_1,\ldots,x_k) = \prod_{i=1}^{k} x_i - \sum_{1\le i<j\le k} x_ix_j.$$

Note that $p_k(\mathbf{x}) = -\binom{m}{2}$, where m is the number of 1s in $\mathbf{x}$, and $\binom{0}{2} = \binom{1}{2} = 0$, unless $m = k$ ($\mathbf{x}$ consists of 1s only), in which case $p_k(\mathbf{x}) = -\binom{m}{2} + 1$.

We claim, that for any $k \ge 4$, the following, denoted by $\mathcal{P}_k$, is a gadget for p_k:

$$p_k(x_1,\ldots,x_k) = \min_{y_0,\ldots,y_{k-4}\in\{0,1\}}\{y_0(3 - 2\sum_{i=1}^{k} x_i) + \sum_{j=1}^{k-4} y_j(2 + j - \sum_{i=1}^{k} x_i)\}.$$

Notice that in the case of $k = 4$, the gadget corresponds to the gadget used in the proof of Theorem 5.6.4, and therefore the base case is proved. We proceed by induction on k. Assume that $\mathcal{P}_i$ is a gadget for p_i for every $i \le k$. We prove that $\mathcal{P}_{k+1}$ is a gadget for p_{k+1}.

Firstly, take the gadget $\mathcal{P}_k$ for p_k, and replace every sum $\sum_{i=1}^{k} x_i$ with $\sum_{i=1}^{k+1} x_i$. We denote the new gadget $\mathcal{P}'$. By the inductive hypothesis, it is not difficult to see that $\mathcal{P}'$ is a valid gadget for p_{k+1} on all assignments with at most $k-1$ 1s. Also, on any assignment with exactly k 1s, $\mathcal{P}'$ returns $-\binom{k}{2} + 1$. On the assignment with $k+1$ 1s, $\mathcal{P}'$ returns: $-\binom{k}{2} + 1 - 2 - 1(k-4)$. This can be simplified as follows: $-\binom{k}{2} + 1 - 2 - k + 4 = -\binom{k}{2} + 1 - k + 2 = -(\binom{k}{2} + \binom{k}{1}) + 1 + 2 = -\binom{k+1}{2} + 1 + 2$.

Hence $\mathcal{P}'$ is *almost* a gadget for p_{k+1}: we only need to subtract 1 on an assignment which has exactly k 1s, and subtract 2 on the assignment consisting of 1s only. But this is exactly what $\min_{y_{k-3}\in\{0,1\}}\{y_{k-3}(2+(k-3)-\sum_{i=1}^{k+1}x_i)\}$ does. Therefore, we have established that $\mathcal{P}_{k+1}$ is a gadget for p_{k+1} over $\Gamma_{\mathsf{sub},2}$ with $k-3$ extra variables.

Given a cost function $\phi \in \Gamma_{\mathsf{suff},k}$, let p be the corresponding polynomial which represents ϕ. By the construction in Theorem 5.5.2, we can replace all negative terms of degree ≥ 3. By the constructions in Theorem 5.5.9 and Theorem 5.6.4, we can replace all positive cubic and quartic terms. Now for any positive term of degree d, $5 \leq d \leq k$, we replace the term with the gadget $\mathcal{P}_d$ and add $\sum_{1\leq i<j\leq k} x_i x_j$ back in. This construction works if all quadratic coefficients of the resulting polynomial are non-positive. However, this is ensured by the definition of $\Gamma_{\mathsf{suff},k}$ and by the choice of the gadgets. □

Corollary 5.6.8. *A polynomial p in n Boolean variables over Γ_{suff} can be minimised in $O((n+r)^3)$ time, where r is the number of cost functions of arity 3 or greater represented in p.*

5.7 Summary

We have studied the expressive power of binary Boolean submodular functions. In the pseudo-Boolean optimisation language, we studied classes of submodular functions whose polynomial representation can be decomposed into a sum of binary submodular polynomials, over a possibly larger set of variables.

We have shown new and simpler proofs of known results for a range of special classes. Moreover, we have shown a new class of submodular functions of arbitrary arities which can be expressed by binary submodular functions. Consequently, we have shown in a uniform way that all these classes of submodular functions can be minimised efficiently by reducing to the (s,t)-Min-Cut problem. The same new class of functions has recently been obtained independently by Zalesky [Zal08] using different gadgets.

Chapter 6 studies the question of which submodular functions are expressible by binary submodular functions in more depth, obtains a more general class of expressible functions and also discusses applications of these results to VCSP instances with submodular constraints.

Related work There has been a lot of research on subclasses of pseudo-Boolean functions which can be minimised[4] using (s,t)-Min-Cut or reduced to this problem by switching a subset of variables (see [BH02] for a recent survey). Note that switching a subset of variables does not preserve submodularity.

In Section 5.5, we discussed the class of *negative-positive* (also known as *almost-positive*) functions. This class has been studied in [Bal70, Rhy70, PR75, PQ82].

[4]Most papers on pseudo-Boolean optimisation deal with the maximisation problem, and therefore talk about supermodular functions, rather than submodular functions. However, the maximisation problem of supermodular functions is equivalent to the minimisation problem of submodular functions.

Functions which can be made negative-positive by switching a subset of variables are called *unate* functions. The class of unate functions has been studied and shown to be recognisable in polynomial time in [SdWC90]. Note that unate functions are not in general submodular.

The class of cost function which can be made submodular by permuting the domains of all variables is called *renamable*, *permutable*, or *switchable* submodular. Testing whether a given binary VCSP instance consists of renamable submodular cost functions can be done in polynomial time for any finite domain [Sch07].

CHAPTER 6

Non-Expressibility of Submodular Functions

A man should look for what is, and not for what he thinks should be.
Albert Einstein (1879–1955)

This chapter is based on the following paper:

[ŽCJ09] S. Živný, D.A. Cohen, and P.G. Jeavons. The Expressive Power of Binary Submodular Functions. *Discrete Applied Mathematics*, 157(15):3347–3358, 2009.
Earlier version in *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science (MFCS'09)*, volume 5734 of *Lecture Notes in Computer Science*, pages 744–757. Springer, 2009.

6.1 Introduction

In Chapter 5, we studied the expressive power of binary submodular functions. We have seen that showing that a class $\mathcal{C}$ of submodular functions is expressible by binary submodular functions is equivalent to showing that functions from $\mathcal{C}$ can be minimised via (s,t)-Min-Cut. Furthermore, showing that a class $\mathcal{C}$ of submodular functions is *not* expressible by binary submodular functions is equivalent to showing that the SFM_b problem for functions from $\mathcal{C}$ *cannot* be reduced to the (s,t)-Min-Cut problem via the expressibility reduction.

It has previously been an open problem whether all Boolean submodular functions can be decomposed into a sum of binary submodular functions over a possibly larger set of variables. This problem has been considered within several different contexts in computer science, including computer vision, artificial intelligence, and pseudo-Boolean optimisation. Using the connection between the expressive power of valued

constraints and certain algebraic properties of functions described in earlier chapters, we answer this question negatively.

6.2 Results

The series of positive expressibility results in Chapter 5 naturally raises the following question:

Problem 6.2.1. *Are* all *submodular polynomials expressible by binary submodular polynomials, over a possibly larger set of variables?*

Each of the above expressibility results was obtained by an ad-hoc construction, and no general technique[1] has previously been proposed which is sufficiently powerful to address Problem 6.2.1.

Using the algebraic approach to characterising the expressive power of valued constraint developed by Cohen et al. in [CCJ06] and discussed in Chapter 3, we are able to give a negative answer to Problem 6.2.1: we show that there exist submodular polynomials of arity 4 that cannot be expressed by binary submodular polynomials. More precisely, we characterise exactly which submodular polynomials of arity 4 are expressible by binary submodular polynomials and which are not. In addition, we show that any submodular polynomial of arity 4 is either expressible by binary submodular polynomials with only a small number of extra variables, or it is not expressible at all.

On the way to establishing these results we show that two broad families of submodular functions, known as *upper fans* and *lower fans*, are all expressible by binary submodular functions. This provides a large new class of submodular polynomials of all arities that are expressible by binary submodular polynomials and hence are solvable efficiently by reduction to (s,t)-Min-Cut. We use the expressibility of this family, and the existence of non-expressible functions, to refute a conjecture from [PY05] on the structure of the extreme rays of the cone of Boolean submodular functions, and suggest a more refined conjecture of our own.

The rest of this chapter is organised as follows. In Section 6.3, we show some new classes of submodular functions to be expressible by binary submodular functions. In particular, we show that all upper fans and lower fans of arbitrary arities are expressible by binary submodular functions. In Section 6.4, we characterise precisely the multimorphisms of all binary submodular functions. Moreover, we characterise the fractional clone of all binary submodular functions, and we present a sufficient condition for a mapping to belong to the multi-clone of all submodular functions. The characterisation of the multi-clone of binary submodular functions helps us to prove, in Section 6.5, the main result of this chapter: there are submodular functions which are *not* expressible by binary submodular functions. Moreover, we characterise precisely which submodular functions of arity 4 can be expressed by binary

[1]For example, standard combinatorial counting techniques cannot resolve this question because we allow arbitrary real-valued coefficients in submodular polynomials. We also allow an arbitrary number of additional variables.

submodular functions. Our results imply limitations on the expressibility reduction and establish for the first time that it cannot be used in general to minimise arbitrary submodular functions. Finally, we refute a conjecture of Promislow and Young [PY05] on the structure of the extreme rays of the cone of Boolean submodular functions. In Section 6.6, we present some results on the recognition problem for submodular functions. Section 6.7 summarises this chapter and comments on related work.

6.3 Expressibility of upper fans and lower fans

We begin by defining some particular families of submodular cost functions, first described in [PY05], which will turn out to play a central role in our analysis.

Definition 6.3.1. Let L be a lattice. We define the following cost functions on L:

- For any set A of pairwise incomparable elements $\{a_1, \ldots, a_m\} \subseteq L$, such that each pair of distinct elements (a_i, a_j) has the same least upper bound, $\bigvee A$, the following cost function is called an *upper fan*:

$$\phi_A(x) \stackrel{\text{def}}{=} \begin{cases} -2 & \text{if } x \geq \bigvee A, \\ -1 & \text{if } x \not\geq \bigvee A, \text{ but } x \geq a_i \text{ for some } i, \\ 0 & \text{otherwise.} \end{cases}$$

- For any set B of pairwise incomparable elements $\{a_1, \ldots, a_m\} \subseteq L$, such that each pair of distinct elements (a_i, a_j) has the same greatest lower bound, $\bigwedge B$, the following cost function is called a *lower fan*:

$$\phi_B(x) \stackrel{\text{def}}{=} \begin{cases} -2 & \text{if } x \leq \bigwedge B, \\ -1 & \text{if } x \not\leq \bigwedge B, \text{ but } x \leq a_i \text{ for some } i, \\ 0 & \text{otherwise.} \end{cases}$$

We call a cost function a *fan* if it is either an upper fan or a lower fan. Note that our definition of fans is slightly more general than the definition in [PY05]. In particular, we allow the set A to be empty, in which case the corresponding upper fan ϕ_A is a constant function. It is not hard to show that all fans are submodular [PY05].

Note that when $D = \{0, 1\}$, the set D^n with the product ordering is isomorphic to the lattice of all subsets of an n-element set ordered by inclusion. Hence, a cost function on a Boolean domain can be viewed as a cost function defined on a lattice of subsets, and we can apply Definition 6.3.1 to identify certain Boolean functions as upper fans or lower fans, as the following example indicates.

Example 6.3.2. Let $A = \{I_1, \ldots, I_r\}$ be a set of subsets of $\{1, 2, \ldots, n\}$ such that for all $i \neq j$ we have $I_i \not\subseteq I_j$ and $I_i \cup I_j = \bigcup A$.

By Definition 6.3.1, the corresponding upper fan function ϕ_A has the following polynomial representation:

$$p(x_1, \ldots, x_n) = (r-2) \prod_{i \in \bigcup A} x_i - \prod_{i \in I_1} x_i - \cdots - \prod_{i \in I_r} x_i.$$

Remark 6.3.3. We remark that any permutation of a set D gives rise to an automorphism of cost functions over D. In particular, for any cost function f on a Boolean domain D, the *dual* of f is the corresponding cost function which results from exchanging the values 0 and 1 for all variables. In other words, if p is the polynomial representation of f, then the dual of f is the cost function whose polynomial representation is obtained from p by replacing all variables x with $1 - x$. Observe that, due to symmetry, taking the dual preserves submodularity and expressibility by binary submodular cost functions.

It follows from Definition 6.3.1 that upper fans are duals of lower fans and vice versa.

Definition 6.3.4. We denote by $\Gamma_{\mathsf{fans},k}$ the set of all fans of arity at most k on a Boolean domain D, and we set $\Gamma_{\mathsf{fans}} \stackrel{\text{def}}{=} \bigcup_k \Gamma_{\mathsf{fans},k}$.

Our next result shows that $\Gamma_{\mathsf{fans}} \subseteq \langle \Gamma_{\mathsf{sub},2} \rangle$; that is, fans of all arities are expressible by binary submodular functions.

Theorem 6.3.5. *Any fan on a Boolean domain D is expressible by binary submodular functions on D using at most $1 + \lfloor m/2 \rfloor$ extra variables, where m is the degree of its polynomial representation.*

Proof. Since upper fans are dual to lower fans, it is sufficient to establish the result for upper fans only.

Let $A = \{I_1, \ldots, I_r\}$ be a set of subsets of $\{1, 2, \ldots, n\}$ such that for all $i \neq j$ we have $I_i \not\subseteq I_j$ and $I_i \cup I_j = \bigcup A$, and let ϕ_A be the corresponding upper fan, as specified by Definition 6.3.1. The polynomial representation of ϕ_A, $p(x_1, \ldots, x_n)$, is given in Example 6.3.2.

The degree of p is equal to the total number of variables occurring in it, which will be denoted m. Note that $m = |\bigcup A|$.

If $r = 0$, then ϕ_A is constant, so the result holds trivially. If $r = 1$, we have $A = \{I\}$, where $I = \{i_1, \ldots, i_m\}$ and the polynomial representation of ϕ_A is $-2x_{i_1}x_{i_2}\cdots x_{i_m}$. In this case, it was shown in Example 5.3.9 that ϕ_A can be expressed by quadratic functions using one extra variable, as follows:

$$-2x_{i_1}x_{i_2}\cdots x_{i_m} = \min_{y \in \{0,1\}} \{2y((m-1) - \sum_{i \in I} x_i)\}.$$

For the case when $r > 1$, we first note that any $i \in \bigcup A$ must belong to all the elements of A except for at most one (otherwise there would be two elements of A, say I_i and I_j, such that $I_i \cup I_j \neq \bigcup A$, which contradicts the choice of A).

We will say that two elements of $\bigcup A$ are *equivalent* if they occur in exactly the same elements of A; that is, $i_1, i_2 \in \bigcup A$ are equivalent if $i_1 \in I_j \Leftrightarrow i_2 \in I_j$ for all $j \in \{i, \ldots, r\}$. Equivalent elements i_1 and i_2 of $\bigcup A$ can be merged by replacing them with a single new element. In the polynomial representation of ϕ_A this corresponds to replacing the variables x_{i_1} and x_{i_2} with a single new variable, z, corresponding to their product. Note that the number of equivalence classes of size two or greater is at most $\lfloor m/2 \rfloor$.

After completing all such merging, we obtain a new set $A' = \{I'_1, \ldots, I'_{r'}\}$ with the property that $|I'_i| = m' - 1$ for every i, where $m' = |\bigcup A'|$ is the size of the common join of any $I'_i, I'_j \in A'$. This set has a corresponding new upper fan, $\phi_{A'}$, over the new merged variables.

To complete the proof we will construct a simple gadget for expressing $\phi_{A'}$, and show how to use this to obtain a gadget for expressing the original upper fan ϕ_A.

Note that the sets I'_i are subsets of $\bigcup A'$, each of size $m' - 1$. Any such subset is uniquely determined by its single missing element. We denote by K the set of elements occurring in *all* sets I'_i and by L the set of elements which are missing from one of these subsets. Clearly, $|K| + |L| = m'$. We claim that the following polynomial is a gadget for expressing ϕ'_A:

$$p'(z_1, \ldots, z_{m'}) = \min_{y \in \{0,1\}} \{y(2(m' - 1) - |L| - \sum_{i \in L} z_i - 2 \sum_{i \in K} z_i)\}.$$

To establish this claim, we will compute the value of p', for each possible assignment to the variables $z_1, \ldots, z_{m'}$. Denote by k_0 the number of 0s assigned to variables in K, and by l_0 the number of 0s assigned to variables in L. Then we have:

$$\begin{aligned} p'(z_1, \ldots, z_{m'}) &= \min_{y \in \{0,1\}} y(2m' - 2 - |L| - \sum_{i \in L} z_i - 2 \sum_{i \in K} z_i) \\ &= \min_{y \in \{0,1\}} y(2m' - 2 - |L| - (|L| - l_0) - 2(m' - |L| - k_0)) \\ &= \min_{y \in \{0,1\}} y(2m' - 2 - 2|L| + l_0 - 2m' + 2|L| + 2k_0) \\ &= \min_{y \in \{0,1\}} y(-2 + 2k_0 + l_0). \end{aligned}$$

Hence if $k_0 = l_0 = 0$, then p' takes the value -2. If $k_0 = 0$ and $l_0 = 1$, then p' takes the value -1. In all other cases (that is, $k_0 > 0$ or $l_0 > 1$), p' takes the value 0. By Definition 6.3.1, this means that p' is the (unique) polynomial representation for $\phi_{A'}$. Note that p' uses just one extra variable, y.

Finally, we show how to obtain a gadget for the original upper fan ϕ_A, from the polynomial p'. Each variable in p' represents an equivalence class of elements of $\bigcup A$, so it can be replaced by a term consisting of the product of the variables in this equivalence class. In this way we obtain a new polynomial over the original variables containing linear and negative quadratic terms together with negative higher-order terms (cubic or above) corresponding to every equivalence class with 2 or more elements. However, each of these higher-order terms can itself be expressed by

a quadratic submodular polynomial, by introducing a single extra variable, as shown in the case when $r = 1$, above. Therefore, combining each of these polynomials, the total number of new variables introduced is at most $1 + \lfloor m/2 \rfloor$. □

All of the expressibility results from Chapter 5 can be obtained as simple corollaries of Theorem 6.3.5, as the following examples indicate. In other words, Theorem 6.3.5 generalises all previously known classes of submodular functions which can be expressed by binary submodular functions.

Example 6.3.6. Any negative monomial $-x_1x_2\cdots x_m$ is a positive multiple of an upper fan, and the positive linear monomial x_1 is equal to $-(1-x_1)+1$, so it is a positive multiple of a lower fan, plus a constant. Hence all negative-positive submodular polynomials are contained in $\mathsf{Cone}(\Gamma_{\mathsf{fans}})$, and by Theorem 6.3.5, they are expressible by binary submodular polynomials, as originally shown in [Rhy70], and also in Theorem 5.5.2.

Example 6.3.7. Any cubic submodular polynomial can be expressed as a positive sum of upper fans [PY05]. Hence, by Theorem 6.3.5, all cubic submodular polynomials are expressible by binary submodular polynomials, as originally shown in [BM85], and also in Theorem 5.5.9.

Example 6.3.8. Recall from Definition 5.5.5 that a Boolean cost function ϕ is called *2-monotone* [CKS01] if there exist two sets $R, S \subseteq \{1,\ldots,n\}$ such that $\phi(\mathbf{x}) = 0$ if $R \subseteq \mathbf{x}$ or $\mathbf{x} \subseteq S$ and $\phi(\mathbf{x}) = 1$ otherwise (where $R \subseteq \mathbf{x}$ means $\forall i \in R, x[i] = 1$ and $\mathbf{x} \subseteq S$ means $\forall i \notin S, x[i] = 0$). It was shown in [CCJK05, Proposition 2.9] that a 2-valued Boolean cost function is 2-monotone if, and only if, it is submodular.

For any 2-monotone cost function defined by the sets of indices R and S, it is straightforward to check that $\phi = \min_{y\in\{0,1\}} y(1+\phi_A/2) + (1-y)(1+\phi_B/2)$ where ϕ_A is the upper fan defined by $A = \{R\}$ and ϕ_B is the lower fan defined by $B = \{\overline{S}\}$. Note that the function $y\phi_A$ is an upper fan, and the function $(1-y)\phi_B$ is a lower fan. Hence, by Theorem 6.3.5, all 2-monotone polynomials are expressible by binary submodular polynomials, and solvable by reduction to (s,t)-Min-Cut, as originally shown in [CKS01], and also in Theorem 5.5.6.

Example 6.3.9. A much studied subclass of submodular functions which can be minimised via (s,t)-Min-Cut is the class of *polar* (also known as *homogeneous* [BM85]) functions [Cra89]. Polar functions are functions which have a posiform representation (this means that all coefficients except the constant one are non-negative) such that all monomials consists of only variables or negated variables [BM85].

More formally, a polynomial is called polar if it can be expressed as a sum of terms of the form $ax_1x_2\ldots x_k$ or $a(1-x_1)(1-x_2)\ldots(1-x_k)$ with positive coefficients a, together with a constant term. It was observed in [BM85] that all polar polynomials are supermodular. Hence in our context it makes sense to talk about negated polar polynomials, which are required to have all coefficients except the constant one non-positive. It follows from results in [BM85] that all negated polar polynomials are submodular.

It is known that for binary and ternary functions, negated polar functions are precisely submodular functions [Cra89]. Moreover, it is known that for functions of arity 4, negated polar functions are strictly included in the class of submodular functions of arity 4 [BM85].

As every negated term $-ax_1x_2\dots x_k$, is a positive multiple of an upper fan, and every negated term $-a(1-x_1)(1-x_2)\dots(1-x_k)$, is a positive multiple of a lower fan, by Theorem 6.3.5, all cost functions which are the negations of polar polynomials are expressible by binary submodular polynomials, and solvable by reduction to (s,t)-Min-Cut, as originally shown in [BM85].

However, Theorem 6.3.5 also provides many new functions of all arities which have not previously been shown to be expressible by binary submodular functions, as the following example indicates.

Example 6.3.10. The function $2x_1x_2x_3x_4-x_1x_2x_3-x_1x_2x_4-x_1x_3x_4-x_2x_3x_4$ belongs to $\Gamma_{\mathsf{fans},4}$, but does not belong to any class of submodular functions which has previously been shown to be expressible by binary submodular functions. In particular, it does not belong to the class Γ_{new} identified in Chapter 5 (see Definition 5.6.3).

6.4 Characterisation of $\mathsf{Mul}(\Gamma_{\mathsf{sub},2})$ and $\mathsf{fPol}(\Gamma_{\mathsf{sub},2})$

Since we have seen that a cost function can only be expressed by a given set of cost functions if it has the same multimorphisms, we now investigate the multimorphisms and fractional polymorphisms of $\Gamma_{\mathsf{sub},2}$.

Notation 6.4.1. A function $\mathcal{F} : D^k \to D^k$ is called *conservative* if, for each possible choice of $x_1,\dots,x_k$, the tuple $\mathcal{F}(x_1,\dots,x_k)$ is a permutation of $x_1,\dots,x_k$ (though different inputs may be permuted in different ways).

Lemma 6.4.2. *Let Γ be a valued constraint language including all unary cost functions. Then any multimorphism $\mathcal{F}$ of Γ, $\mathcal{F} \in \mathsf{Mul}(\Gamma)$, is conservative.*

Proof. Recall from Example 2.1.15 that for any $d \in D$ and $c \in \overline{\mathbb{R}}_+$, we define the unary cost function μ_c^d as follows:

$$\mu_c^d(x) \stackrel{\text{def}}{=} \begin{cases} c & \text{if } x = d, \\ 0 & \text{if } x \neq d. \end{cases}$$

Let $\mathcal{F} : D^k \to D^k$ be a non-conservative function. In that case, there are $u_1,\dots,u_k,v_1,\dots,v_k \in D$ such that $\mathcal{F}(u_1,\dots,u_k) = \langle v_1,\dots,v_k\rangle$ and there is i such that v_i occurs more often in $\langle v_1,\dots,v_k\rangle$ than in $\langle u_1,\dots,u_k\rangle$. But then $\mathcal{F}$ is not a multimorphism of the unary cost function $\mu_1^{v_i}$. Hence any $\mathcal{F} \in \mathsf{Mul}(\Gamma)$ must be conservative.

□

Notation 6.4.3. For any two tuples $\mathbf{x} = \langle x_1,\dots,x_k\rangle$ and $\mathbf{y} = \langle y_1,\dots,y_k\rangle$ over D, we denote by $H(\mathbf{x},\mathbf{y})$ the *Hamming distance* between $\mathbf{x}$ and $\mathbf{y}$, which is the number of positions at which the corresponding values are different.

Definition 6.4.4. We denote by $\Gamma^{\infty}_{\mathsf{sub},2}$ the set of binary submodular cost functions taking finite or infinite costs.

Theorem 6.4.5. *For any Boolean domain D, and any $\mathcal{F} : D^k \to D^k$, the following are equivalent:*

1. $\mathcal{F} \in \mathsf{Mul}(\Gamma_{\mathsf{sub},2})$.
2. $\mathcal{F} \in \mathsf{Mul}(\Gamma^{\infty}_{\mathsf{sub},2})$.
3. *$\mathcal{F}$ is conservative and Hamming distance non-increasing.*

Proof. First we consider unary cost functions. All unary cost functions on a Boolean domain are easily shown to be submodular. Hence, by Lemma 6.4.2, all multimorphisms of $\Gamma_{\mathsf{sub},2}$ and $\Gamma^{\infty}_{\mathsf{sub},2}$ are conservative. On the other hand, any conservative function $\mathcal{F} : D^k \to D^k$ is clearly a multimorphism of any unary cost function, since it merely permutes its arguments.

For any $c \in \overline{\mathbb{R}}_+$, define the binary cost functions λ_c and χ_c as follows:

$$\lambda_c(x,y) \stackrel{\text{def}}{=} \begin{cases} c & \text{if } x = 0 \text{ and } y = 1, \\ 0 & \text{otherwise.} \end{cases} \qquad \chi_c(x,y) \stackrel{\text{def}}{=} \begin{cases} c & \text{if } x \neq y, \\ 0 & \text{otherwise.} \end{cases}$$

(We have seen λ_c in Example 2.1.15.) Note that $\chi_c(x,y) = (\lambda_c(x,y) + \lambda_c(y,x))/2$.

By a simple case analysis, it is straightforward to check that any binary submodular cost function on a Boolean domain can be expressed by binary functions of the form λ_c, with $c > 0$ together with unary cost functions of the form μ^d_c.

We observe that when $c < \infty$, $\lambda_c(x,y) = (\chi_c(x,y) + \mu^0_c(x) + \mu^1_c(y) - c)/2$, so λ_c can be expressed by functions of the form χ_c together with unary cost functions of the form μ^d_c. Hence, since expressibility preserves multimorphisms, $\mathsf{Mul}(\Gamma_{\mathsf{sub},2}) = \mathsf{Mul}(\{\chi_c \mid c \in \mathbb{R}_+, c > 0\}) \cap \mathsf{Mul}(\{\mu^d_c \mid c \in \mathbb{R}_+, d \in D\})$.

Now let $\mathbf{u}, \mathbf{v} \in D^k$, and consider the multimorphism inequality, as given in Definition 2.4.11, for the case where $t_i = \langle \mathbf{u}[i], \mathbf{v}[i] \rangle$, for $i = 1, \ldots, k$. By Definition 2.4.11, for any $c > 0$, $\mathcal{F}$ is a multimorphism of χ_c if, and only if, the following holds for all choices of $\mathbf{u}$ and $\mathbf{v}$:

$$H(\mathbf{u}, \mathbf{v}) \geq H(\mathcal{F}(\mathbf{u}), \mathcal{F}(\mathbf{v})).$$

This proves that the multimorphisms of $\Gamma_{\mathsf{sub},2}$ are precisely the conservative functions which are also Hamming distance non-increasing.

Since $\Gamma_{\mathsf{sub},2} \subseteq \Gamma^{\infty}_{\mathsf{sub},2}$, we know that $\mathsf{Mul}(\Gamma^{\infty}_{\mathsf{sub},2}) \subseteq \mathsf{Mul}(\Gamma_{\mathsf{sub},2})$. Therefore, in order to complete the proof it is enough to show that every conservative and Hamming distance non-increasing function $\mathcal{F}$ is a multimorphism of λ_∞ and χ_∞. However, as χ_∞ is expressible by λ_∞ because $\chi_\infty(x,y) = \lambda_\infty(x,y) + \lambda_\infty(y,x)$, and expressibility preserves multimorphisms, it is enough to show that every conservative and Hamming distance non-increasing function $\mathcal{F}$ is a multimorphism of λ_∞.

For any $\mathbf{u}, \mathbf{v} \in \{0,1\}^k$, the Hamming distance $H(\mathbf{u}, \mathbf{v})$ is equal to the symmetric difference of the sets of positions where $\mathbf{u}$ and $\mathbf{v}$ take the value 1. Hence, for tuples

$\mathbf{u}$ and $\mathbf{v}$ containing some fixed number of 1s, the minimum Hamming distance occurs precisely when one of these sets of positions is contained in the other.

Now consider again the multimorphism inequality, as given in Definition 2.4.11, for the case where $t_i = \langle \mathbf{u}[i], \mathbf{v}[i] \rangle$, for $i = 1, \ldots, k$. If there is any position i where $\mathbf{u}[i] = 0$ and $\mathbf{v}[i] = 1$, then $\lambda_\infty(t_i) = \infty$, so the multimorphism inequality is trivially satisfied. If there is no such position, then the set of positions where $\mathbf{v}$ takes the value 1 is contained in the set of positions where $\mathbf{u}$ takes the value 1, so $H(\mathbf{u}, \mathbf{v})$ takes its minimum possible value over all reorderings of $\mathbf{u}$ and $\mathbf{v}$. Hence if $\mathcal{F}$ is conservative, then $H(\mathbf{u}, \mathbf{v}) \leq H(\mathcal{F}(\mathbf{u}), \mathcal{F}(\mathbf{v}))$, and if $\mathcal{F}$ is Hamming distance non-increasing, we have $H(\mathbf{u}, \mathbf{v}) = H(\mathcal{F}(\mathbf{u}), \mathcal{F}(\mathbf{v}))$. But this implies that the set of positions where $\mathcal{F}(\mathbf{v})$ takes the value 1 is contained in the set of positions where $\mathcal{F}(\mathbf{u})$ takes the value 1. By definition of λ_∞, this implies that both sides of the multimorphism inequality are zero, so $\mathcal{F}$ is a multimorphism of λ_∞. □

Remark 6.4.6. In recent work with Dave Cohen [CŽ09], we have obtained an alternative characterisation of $\mathsf{Mul}(\Gamma_{\mathsf{sub},2})$ as those functions which are strictly subset preserving.

We now show that fractional polymorphisms of binary submodular cost functions have to be fractionally conservative.

Notation 6.4.7. Let $\mathcal{F} = \{\langle r_1, f_1 \rangle, \ldots, \langle r_n, f_n \rangle\}$ be a k-ary weighted mapping, where each r_i is a positive rational number such that $\sum_{i=1}^{n} r_i = k$ and each f_i is a distinct function from D^k to D. Then $\mathcal{F}$ is called *fractionally conservative* if for each possible choice of $x_1, \ldots, x_k$ and every $1 \leq i \leq k$,

$$\sum_{\{j | f_j(x_1, \ldots, x_k) = x_i\}} r_j = 1. \tag{6.1}$$

Lemma 6.4.8. *Let Γ be a valued constraint language including all unary cost functions. Then any fractional polymorphism $\mathcal{F}$ of Γ, $\mathcal{F} \in \mathsf{fPol}(\Gamma)$, is fractionally conservative.*

Proof. Recall from Example 2.1.15 and from the proof of Lemma 6.4.2 that for any $d \in D$ and $c \in \overline{\mathbb{R}}_+$, we define the unary cost function μ_c^d as follows:

$$\mu_c^d(x) \stackrel{\text{def}}{=} \begin{cases} c & \text{if } x = d, \\ 0 & \text{if } x \neq d. \end{cases}$$

First notice that for every $x_1, \ldots, x_k$ and $1 \leq i \leq n$, $f_i(x_1, \ldots, x_k) \in \{x_1, \ldots, x_k\}$. (This is a weaker condition than $\mathcal{F}$ being conservative.) Otherwise, $\mathcal{F}$ cannot be a fractional polymorphisms of the unary cost function $\mu_1^{f_i(x_1, \ldots, x_k)}$.

Now assume that there is an $1 \leq i \leq k$ such that $\sum_{\{j | f_j(x_1, \ldots, x_k) = x_i\}} r_j > 1$. Then clearly $\mathcal{F}$ cannot be a fractional polymorphism of the unary cost function $\mu_1^{x_i}$. Therefore, for all i, $\sum_{\{j | f_j(x_1, \ldots, x_k) = x_i\}} r_j \leq 1$.

Equality 6.1 now follows from the fact that for every $x_1, \ldots, x_k$,

$$\sum_{i=1}^{n} r_i = \sum_{i=1}^{k} \sum_{\{j | f_j(x_1,\ldots,x_k)=x_i\}} r_j = k.$$

□

We now prove an extension of Theorem 6.4.5 to fractional polymorphisms, and hence characterise the fractional clone of binary submodular cost functions.

Notation 6.4.9. For any two tuples $\mathbf{x} = \langle x_1, \ldots, x_k \rangle$ and $\mathbf{y} = \langle y_1, \ldots, y_k \rangle$ over D, and a weight vector $w = \langle w_1, \ldots, w_k \rangle$, we denote by $H(\mathbf{x}, \mathbf{y}, w)$ the *weighted Hamming distance* between $\mathbf{x}$ and $\mathbf{y}$, which is the sum of all w_i such that $x_i \neq y_i$.

Theorem 6.4.10. *For any Boolean domain D, and any k-ary weighted mapping $\mathcal{F} = \{\langle r_1, f_1 \rangle, \ldots, \langle r_n, f_n \rangle\}$, where each r_i is a positive rational number such that $\sum_{i=1}^{n} r_k = k$ and each f_i is a distinct function from D^k to D, the following are equivalent:*

1. $\mathcal{F} \in \mathsf{fPol}(\Gamma_{\mathsf{sub},2})$.
2. $\mathcal{F} \in \mathsf{fPol}(\Gamma^{\infty}_{\mathsf{sub},2})$.
3. *$\mathcal{F}$ is fractionally conservative and weighted Hamming distance non-increasing; that is, for any $\mathbf{x}$ and $\mathbf{y}$, $H(\mathbf{x}, \mathbf{y}) \geq H(\mathcal{F}(\mathbf{x}), \mathcal{F}(\mathbf{y}), w)$, where $w = \langle r_1, \ldots, r_n \rangle$.*

Proof. Similar to the proof of Theorem 6.4.5. First we consider unary cost functions. All unary cost functions on a Boolean domain are easily shown to be submodular. By Lemma 6.4.8, all fractional polymorphisms of $\Gamma_{\mathsf{sub},2}$ and $\Gamma^{\infty}_{\mathsf{sub},2}$ are fractionally conservative. On the other hand, any fractionally conservative weighted mapping $\mathcal{F}$ is clearly a fractional polymorphism of any unary cost function.

Recall from the proof of Theorem 6.4.5 the definition of the cost functions λ_c and χ_c for any $c \in \overline{\mathbb{R}}_+$. We have seen in the proof of Theorem 6.4.5 that $\chi_c(x, y) = (\lambda_c(x, y) + \lambda_c(y, x))/2$, and also that any binary submodular cost function on a Boolean domain can be expressed by binary functions of the form λ_c, with $c \in \overline{\mathbb{R}}_+$ together with unary cost functions of the form μ^d_c.

For $c < \infty$, $\lambda_c(x, y) = (\chi_c(x, y) + \mu^0_c(x) + \mu^1_c(y) - c)/2$, so λ_c can be expressed by functions of the form χ_c together with unary cost functions of the form μ^d_c. Hence, since expressibility preserves fractional polymorphisms, $\mathsf{fPol}(\Gamma_{\mathsf{sub},2}) = \mathsf{fPol}(\{\chi_c \mid c \in \mathbb{R}_+, c > 0\}) \cap \mathsf{fPol}(\{\mu^d_c \mid c \in \mathbb{R}_+, d \in D\})$.

Now let $\mathbf{u}, \mathbf{v} \in D^k$, and consider the fractional polymorphism Inequality 2.1, as given in Definition 2.4.10, for the case where $t_i = \langle \mathbf{u}[i], \mathbf{v}[i] \rangle$, for $i = 1, \ldots, k$. Let $p = H(\mathbf{u}, \mathbf{v})$, and let $q = H(\mathbf{u}, \mathbf{v}, w)$, where $w = \langle r_1, \ldots, r_n \rangle$. By Definition 2.4.10, $\mathcal{F}$ is a fractional polymorphism of χ_c if, and only if, $p \geq q$.

This proves that the fractional polymorphisms of $\Gamma_{\mathsf{sub},2}$ are precisely the fractionally conservative weighted mappings which are weighted Hamming distance non-increasing.

Since $\Gamma_{\mathsf{sub},2} \subseteq \Gamma^{\infty}_{\mathsf{sub},2}$, we know that $\mathsf{fPol}(\Gamma^{\infty}_{\mathsf{sub},2}) \subseteq \mathsf{fPol}(\Gamma_{\mathsf{sub},2})$. Similarly to the argument in the proof of Theorem 6.4.5, in order to complete the proof it is enough to show that every weighted mapping which is fractionally conservative and weighted Hamming distance non-increasing is a fractional polymorphism of λ_∞.

Let $\mathbf{u}, \mathbf{v} \in D^k$. We denote $t_i = \langle \mathbf{u}[i], \mathbf{v}[i] \rangle$, for $i = 1, \ldots, k$, and we denote $t'_i = \langle \mathcal{F}(\mathbf{u})[i], \mathcal{F}(\mathbf{v})[i] \rangle$, for $i = 1, \ldots, n$. If there is any position i where $\mathbf{u}[i] = 0$ and $\mathbf{v}[i] = 1$, then $\lambda_\infty(t_i) = \infty$, and the fractional polymorphism Inequality 2.1, as given in Definition 2.4.10, is trivially satisfied. Hence we can assume that there is no such position. We denote by x_{00} the number of $\langle 0, 0 \rangle$ tuples among t_i, $1 \leq i \leq k$, and similarly for x_{01}, x_{10}, and x_{11}. (Note that by our assumption, $x_{01} = 0$.) We denote by w_{00} the sum of weights w_i where $t'_i = \langle 0, 0 \rangle$, and similarly for w_{01}, w_{10}, and w_{11}.

Since $\mathcal{F}$ is fractionally conservative, we get the following:

$$x_{00} = w_{00} + w_{01} \tag{6.2}$$

$$x_{00} + x_{10} = w_{00} + w_{10} \tag{6.3}$$

Moreover, since $\mathcal{F}$ is weighted Hamming distance non-increasing, we get the following:

$$x_{10} \geq w_{01} + w_{10} \tag{6.4}$$

Equation 6.3 and Inequality 6.4 give

$$w_{00} - x_{00} \geq w_{01} \tag{6.5}$$

Equation 6.2 with Inequality 6.5 give

$$0 \geq w_{01} \tag{6.6}$$

But this means that there are no $\langle 0, 1 \rangle$ tuples among t'_i, $1 \leq i \leq k$. By definition of λ_∞, this implies that both sides of the fractional polymorphism Inequality 2.1, as given in Definition 2.4.10, are zero, so $\mathcal{F}$ is a fractional polymorphism of λ_∞. □

We have just characterised the multi-clone of binary Boolean submodular cost functions $\mathsf{Mul}(\Gamma_{\mathsf{sub},2})$, and also the fractional clone of binary Boolean submodular cost functions $\mathsf{fPol}(\Gamma_{\mathsf{sub},2})$.

We present some results on the multi-clone of all Boolean submodular cost functions $\mathsf{Mul}(\Gamma_{\mathsf{sub}})$. By Lemma 6.4.2, we know that all multimorphisms of Γ_{sub} are conservative. Similarly, by Lemma 6.4.8, we know that all fractional polymorphisms of Γ_{sub} are fractional conservative. We now show a simple sufficient condition for a mapping to be a multimorphism of Γ_{sub}.

Proposition 6.4.11. *If $\mathcal{F}$ can be expressed as a sequence of $\langle \textsc{Min}, \textsc{Max} \rangle$, then $\mathcal{F} \in \mathsf{Mul}(\Gamma_{\mathsf{sub}})$.*

Proof. From the definition of Γ_{sub}, $\langle \textsc{Min}, \textsc{Max} \rangle \in \mathsf{Mul}(\Gamma_{\mathsf{sub}})$. If $\mathcal{F}$ can be expressed as a sequence of $\langle \textsc{Min}, \textsc{Max} \rangle$ (in other words, as a composition of functions which are permuted extensions of $\langle \textsc{Min}, \textsc{Max} \rangle$ as specified in Definition 3.4.12, 3.4.13 and 3.4.14), then $\mathcal{F} \in \mathsf{Mul}(\Gamma_{\mathsf{sub}})$ by Observation 3.4.15. □

Remark 6.4.12. One could expect that any $\mathcal{F} \in \mathsf{Mul}(\Gamma_{\mathsf{sub}})$ can be expressed as a sequence of $\langle \textsc{Min}, \textsc{Max} \rangle$. However, this is not true. Recent work with Dave Cohen [CŽ09] has identified multimorphisms of Γ_{sub} which provably cannot be expressed as a sequence of $\langle \textsc{Min}, \textsc{Max} \rangle$. In more detail, we have shown a class of mappings $\mathcal{F}_k$, $k \geq 4$, such that $\mathcal{F}_k$ cannot be expressed as a sequence of $\langle \textsc{Min}, \textsc{Max} \rangle$, but $\langle \textsc{Min}, \textsc{Max} \rangle \in [\mathcal{F}_k]_m$ for every $k \geq 4$. Moreover, using Theorem 3.4.6, we have shown that $\mathcal{F}_4, \mathcal{F}_5 \in [\langle \textsc{Min}, \textsc{Max} \rangle]_m$, that is, $\mathcal{F}_4, \mathcal{F}_5 \in \mathsf{Mul}(\Gamma_{\mathsf{sub}})$.

6.5 Non-expressibility of Γ_{sub} over $\Gamma_{\mathsf{sub},2}$

Theorem 6.4.5 characterises the multimorphisms of $\Gamma_{\mathsf{sub},2}$, and hence enables us to systematically search (for example, using MATHEMATICA) for multimorphisms of $\Gamma_{\mathsf{sub},2}$ which are not multimorphisms of Γ_{sub}. In this way, we have identified the function $\mathcal{F}_{sep} : \{0,1\}^5 \to \{0,1\}^5$ defined in Figure 6.1. We will show in this section that this function has the remarkable property that it can be used to characterise all the submodular functions of arity 4 which are expressible by binary submodular functions on a Boolean domain. Using this result, we show that some submodular functions are *not* expressible in this way, because they do not have $\mathcal{F}_{sep}$ as a multimorphism.

$\mathbf{x}$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
	0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
	0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
$\mathcal{F}_{sep}(\mathbf{x})$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
	0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
	0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
	0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1

Figure 6.1: Definition of $\mathcal{F}_{sep}$.

Proposition 6.5.1. *$\mathcal{F}_{sep}$ is conservative and Hamming distance non-increasing.*

Proof. Straightforward exhaustive verification. □

Theorem 6.5.2. *For any function $f \in \Gamma_{\mathsf{sub},4}$ the following are equivalent:*

1. $f \in \langle \Gamma_{\mathsf{sub},2} \rangle$.
2. $\mathcal{F}_{sep} \in \mathsf{Mul}(\{f\})$.
3. $f \in \mathsf{Cone}(\Gamma_{\mathsf{fans},4})$.

Proof. First, we show (1) ⇒ (2). Proposition 6.5.1 and Theorem 6.4.5 imply that $\mathcal{F}_{sep}$ is a multimorphism of any binary submodular function on a Boolean domain. Hence having $\mathcal{F}_{sep}$ as a multimorphism is a necessary condition for any submodular cost function on a Boolean domain to be expressible by binary submodular cost functions.

Next, we show (2) ⇒ (3). Consider the complete set of inequalities on the values of a 4-ary cost function resulting from having the multimorphism $\mathcal{F}_{sep}$, as specified in Definition 2.4.11. A routine calculation in MATHEMATICA shows that, out of 16^5 such inequalities, there are 4635 which are distinct. After removing from these all those which are equal to the sum of two others, we obtain a system of just 30 inequalities which must be satisfied by any 4-ary submodular cost function which has the multimorphism $\mathcal{F}_{sep}$. Using the double description method [MRTT53],[2] we obtain from these 30 inequalities an equivalent set of 31 extreme rays which generate the same polyhedral cone of cost functions. These extreme rays all correspond to fans or sums of fans.

Finally, we show (3) ⇒ (1). By Theorem 6.3.5, all fans are expressible over $\Gamma_{\mathsf{sub},2}$. It follows that any cost function in this cone of functions is also expressible over $\Gamma_{\mathsf{sub},2}$. □

Next we show that there are indeed 4-ary submodular cost functions which do not have $\mathcal{F}_{sep}$ as a multimorphism and therefore are not expressible by binary submodular cost functions.

Definition 6.5.3. For any Boolean tuple t of arity 4 containing exactly 2 ones and 2 zeros, we define the 4-ary cost function θ_t as follows:

$$\theta_t(x_1,x_2,x_3,x_4) \stackrel{\text{def}}{=} \begin{cases} -1 & \text{if } (x_1,x_2,x_3,x_4) = (1,1,1,1) \text{ or } (0,0,0,0), \\ 1 & \text{if } (x_1,x_2,x_3,x_4) = t, \\ 0 & \text{otherwise.} \end{cases}$$

Cost functions of the form θ_t have been introduced in [PY05], where they are called *quasi-indecomposable* functions.

Definition 6.5.4. We denote by Γ_{qin} the set of all (six) quasi-indecomposable cost functions of arity 4.

It is straightforward to check that all cost functions from Γ_{qin} are submodular, but the next result shows that they are *not* expressible by binary submodular functions.

Proposition 6.5.5. *For all* $\theta \in \Gamma_{\mathsf{qin}}$, $\mathcal{F}_{sep} \notin \mathsf{Mul}(\{\theta\})$.

Proof. The tableau in Figure 6.2 shows that $\mathcal{F}_{sep} \notin \mathsf{Mul}(\{\theta_{(1,1,0,0)}\})$. Permuting the columns appropriately establishes the result for all other $\theta \in \Gamma_{\mathsf{qin}}$. □

Corollary 6.5.6. *For all* $\theta \in \Gamma_{\mathsf{qin}}$, $\theta \notin \langle \Gamma_{\mathsf{sub},2} \rangle$.

[2] As implemented by the program SKELETON available from `http://www.uic.nnov.ru/~zny/skeleton/`

$$\mathcal{F}_{sep} \quad \begin{array}{cccc} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{array} \quad \begin{array}{c} \\ \\ \xrightarrow{\theta_{(1,1,0,0)}} \\ \\ \\ \\ \\ \\ \\ \xrightarrow{\theta_{(1,1,0,0)}} \\ \\ \\ \end{array} \quad \begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \hline 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{array} \quad \begin{array}{l} \left.\begin{array}{l} \\ \\ \\ \\ \\ \end{array}\right\} \sum = 0 \\ \left.\begin{array}{l} \\ \\ \\ \\ \\ \end{array}\right\} \sum = 1 \end{array}$$

Figure 6.2: $\mathcal{F}_{sep} \notin \mathsf{Mul}(\{\theta_{(1,1,0,0)}\})$.

Proof. By Theorem 6.5.2 and Proposition 6.5.5. □

Are there any other 4-ary submodular cost functions which are not expressible over $\Gamma_{\mathsf{sub},2}$? Promislow and Young characterised the extreme rays of the cone of all 4-ary submodular cost functions and established that $\Gamma_{\mathsf{sub},4} = \mathsf{Cone}(\Gamma_{\mathsf{fans},4} \cup \Gamma_{\mathsf{qin}})$ – see Theorem 5.2 of [PY05]. Hence the results in this section characterise the expressibility of all 4-ary submodular functions.

Promislow and Young conjectured that for $k \neq 4$, all extreme rays of $\Gamma_{\mathsf{sub},k}$ are fans [PY05]; that is, they conjectured that for all $k \neq 4$, $\Gamma_{\mathsf{sub},k} = \mathsf{Cone}(\Gamma_{\mathsf{fans},k})$. However, if this conjecture were true it would imply that all submodular functions of arity 5 and above were expressible by binary submodular functions, by Theorem 6.3.5. This is clearly not the case, because inexpressible cost functions such as those identified in Corollary 6.5.6 can be extended to larger arities (for instance, by adding dummy arguments) and remain inexpressible. Hence our results refute this conjecture for all $k \geq 5$. However, we suggest that this conjecture can be refined to a similar statement concerning just those submodular functions which are expressible by binary submodular functions, as follows:

Conjecture 6.5.7. *For all k, $\Gamma_{\mathsf{sub},k} \cap \langle \Gamma_{\mathsf{sub},2} \rangle = \mathsf{Cone}(\Gamma_{\mathsf{fans},k})$.*

This conjecture was previously known to be true for $k \leq 3$ [PY05]; Theorem 6.3.5 shows that $\mathsf{Cone}(\Gamma_{\mathsf{fans},k}) \subseteq \Gamma_{\mathsf{sub},k} \cap \langle \Gamma_{\mathsf{sub},2} \rangle$ for all k, and Theorem 6.5.2 confirms that equality holds for $k = 4$.

Remark 6.5.8. We have seen in Chapter 4 that in the case of max-closed cost functions there is a difference between finite-valued and general cost functions. Adding infinity makes the hierarchy collapse. By contrast, in the case of submodular cost functions, there is no difference between finite-valued and general cost functions. There are finite-valued submodular cost functions which are not expressible, namely cost functions from Γ_{qin}, but even adding infinite costs to the hidden variables would not help to make these cost functions expressible: by Theorem 2.4.19, expressive power is characterised by fractional polymorphisms; by Theorem 6.4.10, finite-valued and general submodular cost functions have the same fractional polymorphisms.

6.6 The complexity of recognising expressible functions

Finally, we show that we can test efficiently whether a submodular polynomial of arity 4 is expressible by binary submodular polynomials.

Definition 6.6.1. Let $p(x_1, x_2, x_3, x_4)$ be the polynomial representation of a 4-ary submodular cost function f. We denote by a_I the coefficient of the term $\prod_{i \in I} x_i$. We say that f satisfies condition **Sep** if for each $\{i, j\}, \{k, \ell\} \subset \{1, 2, 3, 4\}$, with i, j, k, ℓ distinct, we have $a_{\{i,j\}} + a_{\{k,\ell\}} + a_{\{i,j,k\}} + a_{\{i,j,\ell\}} \leq 0$.

Theorem 6.6.2. *For any $f \in \Gamma_{\mathsf{sub},4}$, the following are equivalent:*

1. *$f \in \langle \Gamma_{\mathsf{sub},2} \rangle$.*
2. *f satisfies condition* **Sep**.

Proof. As in the proof of Theorem 6.5.2, we construct a set of 30 inequalities corresponding to the multimorphism $\mathcal{F}_{sep}$. Each of these inequalities on the values of a cost function can be translated into inequalities on the coefficients of the corresponding polynomial representation by a straightforward linear transformation. This calculation shows that 24 of the resulting inequalities impose the condition of submodularity, and the remaining 6 impose condition **Sep**. Hence a submodular cost function of arity 4 has the multimorphism $\mathcal{F}_{sep}$ if, and only if, its polynomial representation satisfies condition **Sep**. The result then follows from Theorem 6.5.2. □

Using Theorem 6.6.2, we can test whether optimisation problems given as a sum of submodular functions of arity 4 can be reduced to the (s, t)-Min-Cut problem via the expressibility reduction. (These problems arise in Computer Vision and in Valued Constraint Satisfaction Problems and will be mentioned in Section 6.7.)

Furthermore, by Theorem 6.3.5, the number of extra variables needed in this reduction is rather small compared to the theoretical upper bound given in Proposition 3.3.1.

It is known that the problem of recognising whether an arbitrary degree-4 polynomial is submodular is co-NP-complete [Cra89, GS88]. One might hope that the more restricted class of submodular polynomials expressible by binary submodular polynomials would be recognisable in polynomial time. At the moment, the complexity of the recognition problem for submodular polynomials of degree 4 that are expressible by binary submodular polynomials is open.

Remark 6.6.3. Multimorphism $\mathcal{F}_{sep}$ could be used to test whether a given polynomial p of degree 4 is expressible by binary submodular polynomials (and therefore submodular). However, the only known way of testing whether $\mathcal{F}_{sep}$ is a multimorphisms of a given polynomial p (in n variables) is via testing all possible tableaux; this would take exponential time in n.

Remark 6.6.4. Martin Cooper noticed that an easier problem of testing whether a polynomial (in n variables) of degree 4 can be written as the sum of 4-ary submodular polynomials (that is, without any extra variables) can be done in polynomial time just by solving a system of linear equations.

Consequently, assuming P $\neq$ co-NP, there are submodular polynomials of degree 4 which cannot be written as the sum of 4-ary submodular polynomials, and hence need extra variables to be expressible. However, this is not surprising: we have seen an example of such a polynomial in Example 5.6.2.

6.7 Summary

In this section, we have extended our study of the expressive power of binary submodular cost functions. We showed a new class of submodular cost functions, the so-called *fans*, which are expressible by binary submodular cost functions. We also showed that there are submodular cost functions which are not expressible by binary submodular cost functions, and hence are not minimisable by reducing to (s,t)-Min-Cut via the expressibility reduction. Moreover, we characterised precisely which submodular cost functions of arity 4 can be expressed by binary submodular cost functions. We also presented results on the recognition problem, and some more results on the algebraic properties of submodular cost functions. We finish this chapter with applications of our results, and remarks on related work.

Applications to artificial intelligence As mentioned in Chapter 5, any Boolean cost function of arity k can be represented uniquely as a Boolean polynomial. Moreover, if Γ is a set of cost functions on a Boolean domain, with arity at most k, then any instance of $\mathsf{VCSP}(\Gamma)$ with n variables can be uniquely represented as a polynomial p in n Boolean variables, of degree at most k. Conversely, any such polynomial represents an n-ary cost function which can be expressed over a set of cost functions on a Boolean domain, with arity at most k. Note also that over a Boolean domain we have that $x^2 = x$, so p has at most 2^n terms: these correspond to subsets of variables.

Example 6.7.1. Any unary cost function ϕ on a Boolean domain can be expressed as the polynomial $p(x_1) = \phi(0) + (\phi(1) - \phi(0))x_1$. Similarly, any binary cost function ϕ can be expressed as

$$\begin{aligned} p(x_1, x_2) = \phi(0,0) \\ &+ (\phi(1,0) - \phi(0,0))x_1 \\ &+ (\phi(0,1) - \phi(0,0))x_2 \\ &+ (\phi(1,1) - \phi(0,1) - \phi(1,0) + \phi(0,0))x_1x_2. \end{aligned}$$

Consider the valued constraint language Γ from Example 2.2.3. It is easy to check that all of the cost functions in Γ are submodular: unary cost functions are submodular by definition; each binary $\phi \in \Gamma$ satisfies $\phi(0,0) + \phi(1,1) \leq \phi(0,1) + \phi(1,0)$. Hence the instance $\mathcal{P}$ from Example 2.2.3 is an instance of $\mathsf{VCSP}(\Gamma_{\mathsf{sub},2})$. The

corresponding polynomial is

$$\begin{aligned} p(x_1,\ldots,x_5) = 3 &+ 0x_1 - x_2 - x_1x_2 \\ &+ 0 + 2x_1 + 4x_4 - x_1x_4 \\ &+ 0 + 0x_2 + x_3 - x_2x_3 \\ &+ 9 - x_3 - 2x_4 - 5x_3x_4 \\ &+ 3 + x_3 + 2x_5 - 2x_3x_5 \\ &+ 4 - 2x_4 - x_5 + 0x_4x_5 \\ &+ 0 + 5x_2 \\ &+ 4 - 2x_5, \end{aligned}$$

which can be simplified to give

$$\begin{aligned} p(x_1,\ldots,x_5) = 23 &+ 2x_1 + 4x_2 + x_3 - x_5 \\ &- x_1x_2 - x_1x_4 - x_2x_3 - 5x_3x_4 - 2x_3x_5. \end{aligned}$$

The fact that $\mathcal{P}$ is an instance of $\mathsf{VCSP}(\Gamma_{\mathsf{sub},2})$ can be easily seen from the polynomial representation: the polynomial p has all quadratic coefficients non-positive, and hence is submodular.

We can rewrite p as in the proof of Theorem 5.4.1 as follows:

$$\begin{aligned} p(x_1,\ldots,x_5) &= 23 + 2x_1 + 4x_2 + x_3 - x_5 \\ &\quad + (1-x_1)x_2 - x_2 + (1-x_1)x_4 - x_4 + (1-x_2)x_3 - x_3 \\ &\quad + 5(1-x_3)x_4 - 5x_4 + 2(1-x_3)x_5 - 2x_5 \\ &= 23 + 2x_1 + 3x_2 - 6x_4 - 3x_5 \\ &\quad + (1-x_1)x_2 + (1-x_1)x_4 + (1-x_2)x_3 + 5(1-x_3)x_4 + 2(1-x_3)x_5 \\ &= 14 + 2x_1 + 3x_2 + 6(1-x_4) + 3(1-x_5) \\ &\quad + (1-x_1)x_2 + (1-x_1)x_4 + (1-x_2)x_3 + 5(1-x_3)x_4 + 2(1-x_3)x_5. \end{aligned}$$

We can now build a graph G with 5 vertices corresponding to variables x_1 through x_5 and two extra vertices s and t and add edges accordingly (see Figure 6.3).

For every assignment v of values 0 and 1 to variables $x_1,\ldots,x5$, $p(v(x_1),\ldots,v(x_5))$ is equal to the size of the (s,t)-cut in G given by v plus 14 (for the constant term in the posiform representation of p). The minimum cut in G, with value 2, is the set $\{s,x_1,x_2\}$. Therefore, the assignment $x_1 = x_2 = 0$ and $x_3 = x_4 = x_5 = 1$ minimises the polynomial p with total value 16.

Using the same method as in Example 6.7.1, we obtain:

Corollary 6.7.2 (of Theorem 6.3.5)**.** *For any fixed $k \geq 4$, any $\mathsf{VCSP}(\Gamma_{\mathsf{fans},k})$ instance with n variables and p_i constraints of arity i, $3 \leq i \leq k$, is solvable in $O((n+p)^3)$ time, where $p = \sum_{i=3}^{k} p_i(1 + \lfloor i/2 \rfloor)$.*

Moreover, as shown above,$\mathsf{VCSP}(\Gamma_{\mathsf{fans},4})$ is the *maximal* class in $\mathsf{VCSP}(\Gamma_{\mathsf{sub},4})$ which can be solved by reduction to (s,t)-Min-Cut in this way.

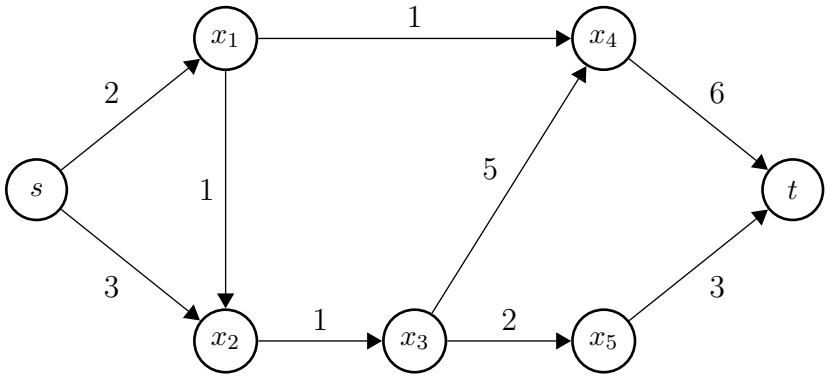

Figure 6.3: Graph G corresponding to polynomial p (Example 6.7.1).

Cohen et al. have shown [CCJ06] that if a cost function ϕ of arity k is expressible by some set of cost functions over Γ, then ϕ is expressible by Γ using at most 2^{2^k} extra variables (Proposition 3.3.1). Theorem 6.3.5 shows that only $O(k)$ extra variables are needed to express any cost function from $\Gamma_{\mathsf{fans},k}$ by $\Gamma_{\mathsf{sub},2}$. Therefore, instances of $\mathsf{VCSP}(\Gamma_{\mathsf{fans}})$ needs fewer extra variables than the theoretical upper bound given by Proposition 3.3.1. In particular, an instance of $\mathsf{VCSP}(\Gamma_{\mathsf{sub},4})$ is either reducible to (s,t)-Min-Cut with only linearly many extra variables,[3] or is not reducible at all.

Applications to computer vision In computer vision, many problems can be naturally formulated in terms of *energy minimisation* where the energy function, over a set of variables $\{x_v\}_{v\in V}$, has the following form:

$$E(\mathbf{x}) \;=\; c_0 + \sum_{v\in V} c_v(x_v) + \sum_{\langle u,v\rangle\in V\times V} c_{uv}(x_u,x_v) + \dots$$

Set V usually corresponds to pixels, x_v denotes the label of pixel $v \in V$ which must belong to some finite domain D. The constant term of the energy is c_0, the unary terms $c_v(\cdot)$ encode data penalty functions, the pairwise and higher-order terms $c_{uv}(\cdot,\cdot)$ and so on are interaction potentials. Functions of arity 3 and above are known as higher-order energy functions, or higher-order cliques. This energy is often derived in the context of Markov Random Fields (also Conditional Random Field) [GG84, Bes86]: a minimum of E corresponds to a *maximum a-posteriori* (MAP) labelling $\mathbf{x}$ [Lau96, WJ08].

As discussed above, there is a direct translation between VCSP instances and pseudo-Boolean polynomials. Hence it is clear that the above mentioned framework of energy optimisation is equivalent to the VCSP. (See [Wer07] for a survey on the connection between computer vision and constraint satisfaction problems, although with a strong emphasis on a linear programming approach.) Therefore, for energy minimisation over Boolean variables we get the following:

[3] Optimal (in the number of extra variables) gadgets for all cost functions from $\Gamma_{\mathsf{fans},4}$ have been identified in [ŽJ08].

Corollary 6.7.3 (of Theorem 6.3.5)**.** *Energy minimisation, where each energy function belongs to* Γ_{fans}*, is solvable in* $O((n+p)^3)$ *time, where* n *is the number of variables (pixels),* p_i *is the number of energy functions of arity* i *and* $p = \sum_{i=3}^{k} p_i(1 + \lfloor i/2 \rfloor)$*.*

Higher-order energy functions have the ability to encode high-level structural dependencies between pixels, which have been shown to be extremely powerful for image labelling problems. They have long been used to model image textures [PL98, RB05, LRHB06], image denoising and restoration [RB05], and also texture segmentation [KKT07]. Their use, however, is severely hampered in practice by the intractable complexity of representing and minimising such functions [RKFJ09]. Our results enlarge the class of higher-order energy functions which can be (exactly) minimised efficiently using graphs cuts. Moreover, despite the theoretical double-exponential upper bound on the number of extra variables (Proposition 3.3.1), the proof of Theorem 6.3.5 shows that any function from Γ_{fans} needs only linearly many (in the arity of the function) extra variables. Hence functions from Γ_{fans} could be used, for instance, in image processing for efficient recognition of images or Bayesian estimation.

Related work We describe an alternative approach to the question of expressibility of submodular cost functions. Cohen et al. have shown that a VCSP instance with binary submodular constraints over a totally-ordered domain can be minimised via (s,t)-Min-Cut in cubic time [CCJK04]. Furthermore, Cohen et al. have shown that a VCSP instance with so-called *2-monotone* constraints over a lattice-ordered domain, which form a subclass of submodular constraints, can be solved via (s,t)-Min-Cut in cubic time. Given a submodular Boolean VCSP instance $\mathcal{P}$ with constraints of arity at most r, one could try to use the *dual representation* [DP89, LD00], which transforms $\mathcal{P}$ into a binary VCSP instance with an exponential blow-up (in r) of the domain size. One could hope to combine these two results and obtain an algorithm for submodular Boolean VCSPs using the (s,t)-Min-Cut problem. An obvious way to complete the above-mentioned construction would be to prove that every binary submodular cost function over a lattice-ordered set can be expressed by 2-monotone cost functions. However, this is not possible as Corollary 6.5.6 gives an example of a binary submodular cost function defined over a lattice-ordered set (in fact, in this case the lattice is distributive) which is not expressible by binary submodular cost functions (and hence not by 2-monotone cost functions).

Open problems Functions which can be made polar by switching a subset of variables are called *unimodular*. The class of unimodular functions was studied in [HS86], and shown to be polynomial-time recognisable in [Cra89]. Note that unimodular functions are not in general submodular. It would be nice to know the precise relationship between submodular cost functions which are expressible by binary submodular cost functions and unimodular cost functions which are submodular. Moreover, what is the biggest subclass of submodular cost functions which is polynomial-time recognisable?

Another interesting open question is the following: is there an infinite hierarchy of submodular cost functions of increasing expressive power (like finite-valued max-

closed cost functions, see Chapter 4) or are all submodular cost functions expressible by a subset of submodular cost functions of a fixed arity?

Is there any characterisation of all separating multimorphisms of expressible submodular cost functions of arity 4 (we have shown just one such multimorphism, namely $\mathcal{F}_{sep}$), or more generally of submodular cost functions of arbitrary arities?

Is Conjecture 6.5.7 true? What is the precise boundary between expressible and non-expressible submodular cost functions?

What is the complexity of the recognition problem for submodular polynomials which are expressible by binary submodular polynomials?

Is there any connection between fans and the general polynomial-time algorithm for the SFM problem by Iwata and Orlin [IO09]?

We have just recently discovered a technical report [SBK00] which uses a computer program to generate all extreme rays of the cone of submodular cost functions of arity 5. Out of 1319 extreme rays, 1172 are not expressible as they do not have $\mathcal{F}_{sep}$ as a multimorphism. This leaves 147 extreme rays. We have not studied yet the expressibility of these cost functions, but these numbers suggest that most submodular cost functions are not expressible by binary submodular cost functions. An immediate question is then: what are the separating multimorphisms for 5-ary submodular cost functions?

Our work deals with Boolean submodular cost functions only. Is there any kind of norm which characterises the algebraic properties of non-Boolean submodular cost functions as Hamming distance does in the case of Boolean submodular cost functions? We have shown that there are non-expressible submodular cost functions of arity 4 over Boolean domains. These results clearly extend to non-Boolean domains. We also know that all ternary submodular cost functions over Boolean domains are expressible by binary submodular cost functions. The only remaining question is whether all ternary submodular cost functions are expressible over 3-element domains. Dave Cohen and Martin Cooper have observed that so-called 3-2-2 submodular cost functions[4] are expressible by binary submodular cost functions.

[4] A 3-2-2 cost function is a ternary cost function whose first argument ranges over a 3-element domain and whose other two arguments range over a 2-element domain.

CHAPTER 7

Summary and Open Problems

In mathematics the art of proposing a question
must be held of higher value than solving it.
Georg Cantor (1845–1918)

7.1 Summary

This thesis has given a detailed examination of the expressive power of valued constraints and related complexity questions. First, we introduced the Valued Constraint Satisfaction problem (VCSP), and provided a long list of many studied optimisation problems that can be naturally described in this framework.

Chapter 3 presented in more detail known results on the expressive power of crisp and valued constraints. We extended these results by showing a new connection between algebraic operations that characterise the expressive power of valued constraints and linear programming. We also started to study systematically the so-called fractional clone theory, and proved a decidability result for the question of whether a given operation belongs to a particular fractional clone.

Chapter 4 considered various classes of valued constraints and the associated cost functions with respect to the question of which of these classes can be expressed using only cost functions of bounded arities. We presented a full classification of various classes of constraints with respect to this problem. We identified the first known example of an infinite chain of classes of constraints with strictly increasing expressive power. Moreover, we characterised the fractional clones of general max-closed cost functions and finite-valued cost functions.

Submodular functions play a key role in combinatorial optimisation and are often considered to be a discrete analogue of convex functions. Both Chapter 5 and Chapter 6 were devoted to investigating the expressive power of binary submodular cost functions.

Chapter 5 presented a new way of looking at the expressibility question of submodular cost functions by binary submodular cost functions. Using our framework, we proved many previously known results in a simple way, and also identified a new class of submodular cost functions of arbitrary arities which can be expressed by binary submodular cost functions. We also explained how the question of expressibility by binary submodular cost functions relates to the (s,t)-Min-Cut problem.

Chapter 6 considered a previously open problem whether all Boolean submodular cost functions can be decomposed into a sum of binary submodular cost functions over a possibly larger set of variables. This problem has been considered within several different contexts in computer science, including computer vision, artificial intelligence, and pseudo-Boolean optimisation. Using a connection between the expressive power of valued constraints and certain algebraic properties of cost functions, we answered this question negatively. Furthermore, we characterised the multi-clone of Boolean binary submodular cost functions and the fractional clone of Boolean binary submodular cost functions.

Our results in Chapter 6 had several corollaries. First, we characterised precisely which submodular polynomials of arity 4 can be expressed by binary submodular polynomials. Next, we identified a novel class of submodular cost functions of arbitrary arities that can be expressed by binary submodular cost functions, and therefore minimised efficiently using a so-called expressibility reduction to the (s,t)-Min-Cut problem. This class generalised all previously known examples. More importantly, our results implied limitations on this kind of reduction and established for the first time that it cannot be used in general to minimise arbitrary submodular cost functions. Moreover, we refuted a conjecture of Promislow and Young on the structure of the extreme rays of the cone of Boolean submodular cost functions. Finally, we also translated our results to other frameworks, namely pseudo-Boolean polynomials and energy minimisation problems from computer vision.

7.2 Open problems

At the end of each chapter, we mentioned related work and open problems. Here, we give a brief list of the most interesting open problems. Some of the problems are related to each other and an answer to one of them could lead to an answer to another one. Problems 1–4 are from Chapter 3, and Problems 5–12 are from Chapter 6.

1. What algebraic operations characterise the expressive power of valued constraints and give rise to a Galois connection between the set of cost functions and the set of operations?

2. What is the structure of fractional clones? What is the closure operator in the set of fractional polymorphisms?

3. Do multimorphisms alone characterise tractability of valued constraints (Conjecture 3.5.4)? Are multimorphisms at least a necessary condition for tractability of valued constraints?

4. Can constant constraints increase the complexity of a valued constraint language? Can adding a constant constraint to a tractable valued constraint language make it intractable?

5. Characterisation of separating multimorphisms of submodular cost functions of arity 4 expressible by binary submodular cost functions.

6. Is there an infinite hierarchy of submodular cost functions of increasing expressive power or are all submodular cost functions expressible by a subset of submodular cost functions of a fixed arity?

7. What is the precise boundary between expressible and non-expressible submodular cost functions? Are expressible exactly those cost functions which lie in the cone of fans (Conjecture 6.5.7)?

8. What is the complexity of the recognition problem for submodular polynomials which are expressible by binary submodular polynomials?

9. Characterisation of the multi-clone of all Boolean submodular cost functions and the fractional clone of all Boolean submodular cost functions.

10. Characterisation of the multi-clone of submodular cost functions and the fractional clone of submodular cost functions over non-Boolean domains.

11. Are all ternary submodular cost functions over 3-element domains expressible by binary submodular cost functions? (And in fact, are all 3-3-2 submodular cost functions expressible by binary submodular cost functions?)

12. Is there any connection between fans and general algorithms for the minimisation problem of submodular functions?

Bibliography

If I have seen further it is by standing on the shoulders of giants.
Sir Isaac Newton (1642–1727)

The numbers after the "→" symbol indicate on which pages each citation occurred.

[ABD07] A. Atserias, A. A. Bulatov, and V. Dalmau. On the Power of k-Consistency. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*, volume 4596 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2007.
doi:10.1007/978-3-540-73420-8_26 → *p. 17*

[ABKW08] T. Achterberg, T. Berthold, T. Koch, and K. Wolter. Constraint Integer Programming: A New Approach to Integrate CP and MIP. In *Proceedings of the 5th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'08)*, volume 5015 of *Lecture Notes in Computer Science*, pages 6–20. Springer, 2008.
doi:10.1007/978-3-540-68155-7_4 → *p. 4*

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999. → *p. 13*

[ACP87] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of Finding an Embedding in k-trees. *SIAM Journal of Algebraic and Discrete Methods*, 8(2):277–284, 1987.
doi:10.1137/0608024 → *p. 14*

[AGG07] I. Adler, G. Gottlob, and M. Grohe. Hypertree width and related hypergraph invariants. *European Journal of Combinatorics*, 28(8):2167–2181, 2007.
doi:10.1016/j.ejc.2007.04.013 → *p. 14*

[AK04] D. Avis and B. Kaluzny. Solving Ineqaualities and Proving Farkas's Lemma Made Easy. *American Mathematical Monthly*, 111(2):152–157, 2004.
http://www.jstor.org/stable/4145216 → *p. 40*

[Apt03] K. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003. → *pp. 2, 32*

[AW09] A. Atserias and M. Weyer. Decidable Relationships between Consistency Notions for Constraint Satisfaction Problems. In *Proceedings of the 18th Annual Conference of the European Association for Computer Science Logic (CSL'09)*, volume 5771 of Lecture Notes in Computer Science, pages 102–116. Springer, 2009.
doi:10.1007/978-3-642-04027-6_10 → *p. 17*

[Bal70] M. Balinksi. On a selection problem. *Management Science*, 17(3):230–231, 1970.
http://www.jstor.org/stable/2629092 → *p. 95*

[BBJK03] F. Börner, A. Bulatov, P. Jeavons, and A. Krokhin. Quantified Constraints: Algorithms and Complexity. In *Proceedings of Computer Science Logic, the 17th Inernational Workshop (CSL'03), the 12th Annual Conference of the EACSL, and the 8th Kurt Gödel Colloquium*, volume 2803 of *Lecture Notes in Computer Science*, pages 58–70. Springer, 2003.
doi:10.1007/b13224 → *p. 32*

[BC07] M. Bodirsky and H. Chen. Quantified Equality Constraints. In *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science (LICS'07)*, pages 203–212. 2007.
doi:10.1109/LICS.2007.38 → *p. 32*

[BC09] M. Bodirsky and H. Chen. Relatively quantified constraint satisfaction. *Constraints*, 14(1):3–15, 2009.
doi:10.1007/s10601-008-9054-z → *p. 32*

[BCRV03] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks, part I: Post's lattice with applications to complexity theory. *ACM SIGACT-Newsletter*, 34(4):38–52, 2003.
doi:10.1145/954092.954101 → *p. 37*

[BCRV04] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks, part II: Constraint Satisfaction Problems. *ACM SIGACT-Newsletter*, 35(1):22–35, 2004.
doi:10.1145/970831.970840 → *p. 37*

[BD06] A. Bulatov and V. Dalmau. A simple algorithm for Mal'tsev constraints. *SIAM Journal on Computing*, 36(1):16–27, 2006.
doi:10.1137/050628957 → *p. 17*

[BD07] A. A. Bulatov and V. Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. *Information and Computation*, 205(5):651–678, 2007.
doi:10.1016/j.ic.2006.09.005 → *p. 21*

[BDG+09] A. Bulatov, M. E. Dyer, L. A. Goldberg, M. Jalsenius, and D. Richerby. The Complexity of Weighted Boolean #CSP with Mixed Signs. *Theoretical Computer Science*, 410(38-40):3949–3961, 2009.
doi:10.1016/j.tcs.2009.06.003 → *p. 21*

[Bes86] J. Besag. On the Statistical Analysis of Dirty Pictures. *Journal of the Royal Statistical Society, Series B*, 48(3):259–302, 1986.
http://www.jstor.org/stable/2345426 → *p. 114*

[BFM+99] 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(3):199–240, 1999.
doi:10.1023/A:1026441215081 → *pp. 9, 11*

[BG05] A. Bulatov and M. Grohe. The complexity of partition functions. *Theoretical Computer Science*, 348(2-3):148–186, 2005.
doi:10.1016/j.tcs.2005.09.011 → *p. 21*

[BG08] M. Bodirsky and M. Grohe. Non-dichotomies in Constraint Satisfaction Complexity. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *Lecture Notes in Computer Science*, pages 184–196. Springer, 2008.
doi:10.1007/978-3-540-70583-3_16 → *p. 32*

[BH02] E. Boros and P. L. Hammer. Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225, 2002.
doi:10.1016/S0166-218X(01)00341-9 → *pp. 31, 62, 83, 86, 88, 89, 91, 95*

[BIM+] J. Berman, P. Idziak, P. Marković, R. McKenzie, M. Valeriote, and R. Willard. Varieties with few subalgebras of powers. To appear in *Transactions of the American Mathematical Society*, 2009. → *p. 17*

[BJHM88] J. Bang-Jensen, P. Hell, and G. MacGillivray. The Complexity of Colouring by Semicomplete Digraphs. *SIAM Journal on Discrete Mathematics*, 1(3):281–298, 1988.
doi:10.1137/0401029 → *p. 19*

[BK08a] M. Bodirsky and J. Kára. The complexity of equality constraint languages. *Theory of Computing Systems*, 43(2):136–158, 2008.
doi:10.1007/s00224-007-9083-9 → *p. 32*

[BK08b] M. Bodirsky and J. Kára. The Complexity of Temporal Constraint Satisfaction Problems. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC'08)*, pages 29–38. 2008.
doi:10.1145/1374376.1374382 → *pp. 4, 32*

[BK09] L. Barto and M. Kozik. Constraint Satisfaction Problems of Bounded Width. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS'09)*. IEEE Computer Society, 2009. → *p. 17*

[BKJ05] A. Bulatov, A. Krokhin, and P. Jeavons. Classifying the Complexity of Constraints using Finite Algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005.
doi:10.1137/S0097539700376676 → *pp. 4, 5, 17, 22, 26, 33, 51*

[BKKR69] V. Bodnarčuk, L. Kalužnin, V. Kotov, and B. Romov. Galois theory for Post algebras. I. *Cybernetics and Systems Analysis*, 5(3):243–252, 1969.
doi:10.1007/BF01070906 → *pp. 26, 34*

[BKL08] A. A. Bulatov, A. Krokhin, and B. Larose. Dualities for Constraint Satisfaction Problems. In *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 93–124. Springer, 2008.
doi:10.1007/978-3-540-92800-3_5 → *p. 17*

[BKMN09] L. Barto, M. Kozik, M. Maróti, and T. Niven. CSP dichotomy for special triads. *Proceedings of the American Mathematical Society*, 137(9):2921–2934, 2009.
doi:10.1090/S0002-9939-09-09883-9 → *p. 17*

[BKN09] L. Barto, M. Kozik, and T. Niven. The CSP dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of Bang-Jensen and Hell). *SIAM Journal on Computing*, 38(5):1782–1802, 2009.
doi:10.1137/070708093 → *p. 17*

[BKR96] R. Burkard, B. Klinz, and R. Rudolf. Perspectives of Monge Properties in Optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996.
doi:10.1016/0166-218X(95)00103-X → *p. 84*

[BM85] A. Billionet and M. Minoux. Maximizing a supermodular pseudo-Boolean function: a polynomial algorithm for cubic functions. *Discrete Applied Mathematics*, 12(1):1–11, 1985.
doi:10.1016/0166-218X(85)90035-6 → *pp. 83, 90, 91, 102, 103*

[BNvO09] M. Bodirsky, G. Nordh, and T. von Oertzen. Integer programming with 2-variable equations and 1-variable inequalities. *Information Processing Letters*, 109(11):572–575, 2009.
doi:10.1016/j.ipl.2009.01.025 → *p. 4*

[Bod96] H. L. Bodlaender. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.
doi:10.1137/S0097539793251219 → *p. 14*

[Bod08] M. Bodirsky. Constraint Satisfaction Problems with Iinfinite Domains. In *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 196–228. Springer, 2008.
doi:10.1007/978-3-540-92800-3_8 → *p. 32*

[BRSV05] E. Böhler, S. Reith, H. Schnoor, and H. Vollmer. Bases for Boolean co-clones. *Information Processing Letters*, 96(2):59–66, 2005.
doi:10.1016/j.ipl.2005.06.003 → *p. 56*

[Bul03] A. A. Bulatov. Tractable Conservative Constraint Satisfaction Problems. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science(LICS'03)*, pages 321–330. IEEE Press, 2003.
doi:10.1109/LICS.2003.1210072 → *pp. 17, 19*

[Bul05] A. A. Bulatov. H-Coloring dichotomy revisited. *Theoretical Computer Science*, 349(1):31–39, 2005.
doi:10.1016/j.tcs.2005.09.028 → *p. 19*

[Bul06] A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006.
doi:10.1145/1120582.1120584 → *p. 17*

[Bul08] A. A. Bulatov. The complexity of the counting constraint satisfaction problem. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *Lecture Notes in Computer Science*, pages 646–661. Springer, 2008.
doi:10.1007/978-3-540-70575-8_53 → *p. 21*

[CCJ06] D. A. Cohen, M. C. Cooper, and P. G. Jeavons. An Algebraic Characterisation of Complexity for Valued Constraints. In *Proceedings of the 12th International Conference on Principles and Practice of Contraint Programming (CP'06)*, volume 4204 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2006.
doi:10.1007/11889205_10
→ *pp. 7, 8, 27, 28, 29, 33, 34, 39, 40, 41, 42, 49, 98, 114*

[CCJ08] D. A. Cohen, M. C. Cooper, and P. G. Jeavons. Generalising submodularity and Horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. *Theoretical Computer Science*, 401(1-3):36–51, 2008.
doi:10.1016/j.tcs.2008.03.015 → *p. 31*

[CCJK04] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. A Maximal Tractable Class of Soft Constraints. *Journal of Artificial Intelligence Research*, 22:1–22, 2004.
doi:10.1613/jair.1400 → *pp. 84, 115*

[CCJK05] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. Supermodular Functions and the Complexity of MAX-CSP. *Discrete Applied Mathematics*, 149(1-3):53–72, 2005.
doi:10.1016/j.dam.2005.03.003 → *pp. 18, 83, 84, 86, 89, 90, 102*

[CCJK06] D. A. Cohen, M. C. Cooper, P. G. Jeavons, and A. A. Krokhin. The Complexity of Soft Constraint Satisfaction. *Artificial Intelligence*, 170(11):983–1016, 2006.
doi:10.1016/j.artint.2006.04.002
→ *pp. 7, 9, 14, 18, 23, 27, 28, 31, 42, 62, 65, 67, 84*

[CCL09] J.-Y. Cai, X. Chen, and P. Lu. Graph Homomorphisms with Complex Values: A Dichotomy Theorem. Technical report, March 2009. ArXiv:0903.4728.
http://arxiv.org/abs/0903.4728 → *p. 21*

[CD05] H. Chen and V. Dalmau. Beyond Hypertree Width: Decomposition Methods Without Decompositions. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, volume 3709 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2005.
doi:/10.1007/11564751_15 → *p. 14*

[CH] Y. Crama and P. L. Hammer. *Boolean Functions - Theory, Algorithms, and Applications*. In preparation. → *pp. 31, 62, 83, 86, 88, 91*

[CH96] N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125(1):1–12, 1996.
doi:10.1006/inco.1996.0016 → *p. 20*

[Che04] H. Chen. *The Computational Complexity of Quantified Constraint Satisfaction.* Ph.D. thesis, Cornell University, 2004. → *p. 32*

[Che06] H. Chen. A rendezvous of logic, complexity, and algebra. *SIGACT News*, 37(4):85–114, 2006.
doi:10.1145/1189056.1189076 → *pp. 17, 22*

[Che08a] H. Chen. The Complexity of Quantified Constraint Satisfaction: Collapsibility, Sink Algebras, and the Three-Element Case. *SIAM Journal on Computing*, 37(5):1674–1701, 2008.
doi:10.1137/060668572 → *p. 32*

[Che08b] H. Chen. Quantified Constraint Satisfaction and the Polynomially Generated Powers Property. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *Lecture Notes in Computer Science*, pages 197–208. Springer, 2008.
doi:10.1007/978-3-540-70583-3_17 → *p. 32*

[Che09] H. Chen. Existentially Restricted Quantified Constraint Satisfaction. *Information and Computation*, 207(3):369–388, 2009.
doi:10.1016/j.ic.2008.11.001 → *p. 32*

[CJ06] D. Cohen and P. Jeavons. The complexity of constraint languages. In F. Rossi, P. van Beek, and T. Walsh, editors, *The Handbook of Constraint Programming.* Elsevier, 2006. → *pp. 5, 17*

[CJG08] D. Cohen, P. Jeavons, and M. Gyssens. A unified theory of structural tractability for constraint satisfaction problems. *Journal of Computer and System Sciences*, 74(5):721–743, 2008.
doi:10.1016/j.jcss.2007.08.001 → *p. 14*

[CJS08] M. C. Cooper, P. G. Jeavons, and A. Z. Salamon. Hybrid tractable CSPs which generalize tree structure. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 530–534. IOS Press, 2008.
doi:10.3233/978-1-58603-891-5-530 → *p. 22*

[CJŽ08] D. A. Cohen, P. G. Jeavons, and S. Živný. The expressive power of valued constraints: Hierarchies and collapses. *Theoretical Computer Science*, 409(1):137–153, 2008.
doi:10.1016/j.tcs.2008.08.036 → *p. 52*

[CKS01] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classification of Boolean Constraint Satisfaction Problems*, volume 7 of *SIAM Monographs on Discrete Mathematics and Applications.* SIAM, 2001. → *pp. 4, 5, 9, 12, 16, 18, 20, 22, 32, 83, 90, 102*

[CKV08] N. Creignou, P. G. Kolaitis, and H. Vollmer, editors. *Complexity of Constraints: An Overview of Current Research Themes*, volume 5250 of *Lecture Notes in Computer Science*. Springer, 2008.
doi:10.1007/978-3-540-92800-3 → *p. 17*

[CLX08] J.-Y. Cai, P. Lu, and M. Xia. Holographic Reduction, Interpolation and Hardness. Unpublished manuscript, 2008. → *p. 21*

[CLX09] J.-Y. Cai, P. Lu, and M. Xia. Holant Problems and Counting CSP. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC'09)*, pages 715–724. 2009.
doi:10.1145/1536414.1536511 → *p. 21*

[Coh03] D. A. Cohen. A New Classs of Binary CSPs for which Arc-Constistency Is a Decision Procedure. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP'03)*, volume 2833 of *Lecture Notes in Computer Science*, pages 807–811. Springer, 2003.
doi:10.1007/b13743 → *p. 22*

[Coo05] M. Cooper. High-order Consistency in Valued Constraint Satisfaction. *Constraints*, 10(3):283–305, 2005.
doi:10.1007/s10601-005-2240-3 → *p. 11*

[Coo08a] M. C. Cooper. Private communication, 2008. → *p. 31*

[Coo08b] M. C. Cooper. Minimization of Locally Defined Submodular Functions by Optimal Soft Arc Consistency. *Constraints*, 13(4):437–458, 2008.
doi:10.1007/s10601-007-9037-5 → *pp. 31, 32*

[Cra89] Y. Crama. Recognition problems for special classes of polynomials in 0-1 variables. *Mathematical Programming*, 44(1-3):139–155, 1989.
doi:10.1007/BF01587085 → *pp. 91, 102, 103, 111, 115*

[Cun84] W. H. Cunningham. Testing membership in matroid polyhedra. *Journal of Combinatorial Theory, Series B*, 36(2):161–188, 1984.
doi:10.1016/0095-8956(84)90023-6 → *p. 30*

[Cun85] W. H. Cunningham. On submodular function minimization. *Combinatorica*, 5(3):185–192, 1985.
doi:10.1007/BF02579361 → *p. 30*

[CŽ09] D. Cohen and S. Živný. Multimorphisms of Submodular Functions. Unpublished manuscript, 2009. → *pp. 105, 108*

[Dal06] V. Dalmau. Generalized Majority-Minority Operations are Tractable. *Logical Methods in Computer Science*, 2(4), 2006.
doi:10.2168/LMCS-2(4:1)2006 → *p. 17*

[Dec90] R. Dechter. On the Expressiveness of Networks with Hidden Variables. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI'90)*, pages 556–562. 1990.
http://www.aaai.org/Library/AAAI/1990/aaai90-084.php → *p. 80*

[Dec92] R. Dechter. From Local to Global Consistency. *Artificial Intelligence*, 55(1):87–107, 1992.
doi:10.1016/0004-3702(92)90043-W → *p. 17*

[Dec03] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003. → *pp. 2, 9, 14, 32*

[DG00] M. E. Dyer and C. S. Greenhill. The complexity of counting graph homomorphisms. *Random Structures and Algorithms*, 17(3-4):260–289, 2000.
doi:10.1002/1098-2418(200010/12)17:3/4<260::AID-RSA5>3.0.CO;2-W → *p. 20*

[DGJ08] M. E. Dyer, L. A. Goldberg, and M. Jerrum. A complexity dichotomy for hypergraph partition functions. Technical report, November 2008. ArXiv:abs/0811.0037.
http://arxiv.org/abs/0811.0037 → *p. 21*

[DGJ09a] M. E. Dyer, L. A. Goldberg, and M. Jerrum. An approximation trichotomy for Boolean #CSP. To appear in *Journal of Computer and System Sciences*, 2009. ArXive:abs/0710.4272.
http://arxiv.org/abs/0710.4272 → *p. 20*

[DGJ09b] M. E. Dyer, L. A. Goldberg, and M. Jerrum. The Complexity of Weighted Boolean #CSP. *SIAM Journal on Computing*, 38(5):1970–1986, 2009.
doi:10.1137/070690201 → *p. 21*

[DGJR09] M. E. Dyer, L. A. Goldberg, M. Jalsenius, and D. Richerby. The Complexity of Approximating Bounded-Degree Boolean #CSP. Technical report, July 2009. ArXive:abs/0907.2663.
http://arxiv.org/abs/0907.2663 → *p. 21*

[DGP07] M. E. Dyer, L. A. Goldberg, and M. Paterson. On counting homomorphisms to directed acyclic graphs. *Journal of the ACM*, 54(6), 2007.
doi:10.1145/1314690.1314691 → *p. 20*

[DJ04] V. Dalmau and P. Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1-3):315–323, 2004.
doi:10.1016/j.tcs.2004.08.008 → *p. 22*

[DJKK08] V. Deineko, P. Jonsson, M. Klasson, and A. Krokhin. The approximability of Max CSP with fixed-value constraints. *Journal of the ACM*, 55(4), 2008.
doi:10.1145/1391289.1391290 → *pp. 18, 31*

[DKV02] V. Dalmau, P. G. Kolaitis, and M. Y. Vardi. Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics. In *Proceedings of the 8th International Conference on Principles and Practice of Contraint Programming (CP'02)*, volume 2470 of *Lecture Notes in Computer Science*, pages 310–326. Springer, 2002.
doi:10.1007/3-540-46135-3_21 → *p. 14*

[DM07] R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.
doi:10.1016/j.artint.2006.11.003 → *p. 10*

[DP89] R. Dechter and J. Pearl. Tree Clustering for Constraint Networks. *Artificial Intelligence*, 38(3):353–366, 1989.
doi:10.1016/0004-3702(89)90037-4 → *pp. 14, 80, 115*

[DP99] V. Dalmau and J. Pearson. Set Functions and Width 1 Problems. In *Proceedings of the 5th International Conference on Constraint Programming (CP'99)*, volume 1713 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 1999.
doi:10.1007/b72297 → *p. 17*

[DW02] K. Denecke and S. Wismath. *Universal Algebra and Applications in Theoretical Computer Science*. Chapman and Hall/CRC Press, 2002. → *pp. 26, 36, 37, 55*

[Edm70] J. Edmonds. Submodular Functions, Matroids, and Certain Polyhedra. *Combinatorial Structures and Their Applications*, pages 69–87, 1970.
doi:10.1007/3-540-36478-1_2 → *p. 30*

[Fea95] A. Fearnley. A strongly rigid binary relation. *Acta Scientarium Mathematicarum (Szeged)*, 61(1-4):35–41, 1995. → *p. 57*

[Fei98] U. Feige. A threshold of ln n for approximating Set Cover. *Journal of the ACM*, 45(4):634–652, 1998.
doi:10.1145/285055.285059 → *p. 30*

[FG06] J. Flum and M. Grohe. *Parametrized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. → *pp. 12, 14*

[FHH03] T. Feder, P. Hell, and J. Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42(1):61–80, 2003.
doi:10.1002/jgt.10073 → *p. 19*

[FI03] L. Fleischer and S. Iwata. A push-relabel framework for submodular function minimization and applications to parametric optimization. *Discrete Applied Mathematics*, 131(2):311–322, 2003.
doi:10.1016/S0166-218X(02)00458-4 → *p. 30*

[FJ07] T. Färnqvist and P. Jonsson. Bounded Tree-Width and CSP-Related Problems. In *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC'07)*, volume 4835 of *Lecture Notes in Computer Science*, pages 632–643. Springer, 2007.
doi:10.1007/978-3-540-77120-3_55 → *p. 19*

[FK06] T. Feder and P. Kolaitis. Closures and dichotomies for quantified constraints. Technical Report TR06-160, Electronic Colloquium on Computational Complexity (ECCC), 2006.
http://eccc.hpi-web.de/eccc-reports/2006/TR06-160/ → *p. 32*

[FMV07] U. Feige, V. S. Mirrokni, and J. Vondrák. Maximizing non-monotone submodular functions. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 461–471. IEEE Computer Society, 2007.
doi:10.1109/FOCS.2007.29 → *pp. 30, 31, 83*

[Fre90] E. C. Freuder. Complexity of K-Tree Structured Constraint Satisfaction Problems. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI'90)*, pages 4–9. 1990.
http://www.aaai.org/Library/AAAI/1990/aaai90-001.php → *p. 14*

[Fuj05] S. Fujishige. *Submodular Functions and Optimization*, volume 58 of *Annals of Discrete Mathematics*. North-Holland, Amsterdam, 2nd edition, 2005. → *p. 30*

[FV98] T. Feder and M. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.
doi:10.1137/S0097539794266766 → *pp. 5, 12, 17, 50*

[GC08] M. J. Green and D. A. Cohen. Domain permutation reduction for constraint satisfaction problems. *Artificial Intelligence*, 172(8-9):1094–1118, 2008.
doi:10.1016/j.artint.2007.12.001 → *pp. 31, 81*

[Gei68] D. Geiger. Closed systems of functions and predicates. *Pacific Journal of Mathematics*, 27(1):95–100, 1968.
http://projecteuclid.org/euclid.pjm/1102985564 → *pp. 26, 34*

[GG84] S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.
doi:10.1109/TPAMI.1984.4767596 → *pp. 31, 114*

[GGJT09] L. A. Goldberg, M. Grohe, M. Jerrum, and M. Thurley. A complexity dichotomy for partition functions with mixed signs. In *Proceedings of the 26th Annual Symposium on Theoretical Aspects of Computer Science (STACS'09)*, pages 493–504. 2009.
http://drops.dagstuhl.de/opus/volltexte/2009/1821 → *p. 21*

[GGK+09] A. Gupta, G. Gutin, M. Karimi, E. Kim, and A. Rafiey. Minimum Cost Homomorphisms to Locally Semicomplete and Quasi-Transitive Digraphs. To appear in *Australasian Journal of Combinatorics*, 2009. → *p. 20*

[GGM+05] G. Gottlob, M. Grohe, N. Musliu, M. Samer, and F. Scarcello. Hypertree Decompositions: Structure, Algorithms, and Applications. In *Proceedings on the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG'05)*, volume 3787 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2005.
doi:10.1007/11604686_1 → *p. 14*

[GGY08] A. R. G. Gutin and A. Yeo. Minimum Cost Homomorphisms to Semicomplete Bipartite Digraphs. *SIAM Journal on Discrete Mathematics*, 22(4):1624–1639, 2008.
doi:10.1137/060668316 → *p. 20*

[GHKR08] A. Gupta, P. Hell, M. Karimi, and A. Rafiey. Minimum Cost Homomorphisms to Reflexive Digraphs. In *Proceedings of the 8th Latin American Symposium on*

Theoretical Informatics (LATIN'08), volume 4957 of *Lecture Notes in Computer Science*, pages 182–193. Springer, 2008.
doi:10.1007/978-3-540-78773-0_16 → *p. 20*

[GHRY08] G. Gutin, P. Hell, A. Rafiey, and A. Yeo. A dichotomy for minimum cost graph homomorphisms. *European Journal of Combinatorics*, 29(4):900–911, 2008.
doi:10.1016/j.ejc.2007.11.012 → *p. 19*

[GHSZ08] A. Gil, M. Hermann, G. Salzer, and B. Zanuttini. Efficient Algorithms for Description Problems over Finite Totally Ordered Domains. *SIAM Journal on Computing*, 38(3):922–945, 2008.
doi:10.1137/050635900 → *p. 55*

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979. → *pp. 16, 57*

[GJ04] R. Gault and P. Jeavons. Implementing a test for tractability. *Constraints*, 9(2):139–160, 2004.
doi:10.1023/B:CONS.0000024049.41091.71 → *p. 35*

[GK07] G. Gutin and E. J. Kim. On the Complexity of the Minimum Cost Homomorphism Problem for Reflexive Multipartite Tournaments. Technical report, 2007. ArXive:abs/0708.2544.
http://arxiv.org/abs/0708.2544 → *p. 20*

[GK08] G. Gutin and E. Kim. Introduction to the Minimum Cost Homomorphism Problem for Directed and Undirected Graphs. *Lectures Notes of the Ramanujan Mathematical Society*, 7:25–37, 2008. → *p. 20*

[GK09] G. Gutin and E. Kim. Complexity of the Minimum Cost Homomorphism Problem for Semicomplete Digraphs with Possible Loops. *Discrete Applied Mathematics*, 2009.
doi:10.1016/j.dam.2009.07.013 → *p. 20*

[GKKR08] A. Gupta, M. Karimi, E. J. Kim, and A. Rafiey. Minimum Cost Homomorphism Dichotomy for Locally In-Semicomplete Digraphs. In *Proceedings of the 2nd International Conference on Combinatorial Optimization and Applications (COCOA'08)*, volume 5165 of *Lecture Notes in Computer Science*, pages 374–383. Springer, 2008.
doi:10.1007/978-3-540-85097-7_35 → *p. 20*

[GKL+07] E. Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Y. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2007. → *p. 17*

[GLS81] M. Grötschel, L. Lovasz, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–198, 1981.
doi:10.1007/BF02579273 → *p. 30*

[GLS88] M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. → *p. 30*

[GLS00] G. Gottlob, L. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.
doi:10.1016/S0004-3702(00)00078-3 → *p. 14*

[GLS02] G. Gottlob, L. Leone, and F. Scarcello. Hypertree decomposition and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.
doi:10.1006/jcss.2001.1809 → *p. 14*

[GM06] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 289–298. 2006.
doi:10.1145/1109557.1109590 → *p. 14*

[GMS07] G. Gottlob, Z. Miklós, and T. Schwentick. Generalized Hypertree Decompositions: NP-hardness and Tractable Variants. In *Proceedings of the 26th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'07)*, pages 13–22. 2007.
doi:10.1145/1265530.1265533 → *p. 14*

[Gro07] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1), 2007.
doi:10.1145/1206035.1206036 → *pp. 14, 19*

[GRY06] G. Gutin, A. Rafiey, and A. Yeo. Minimum Cost and List Homomorphisms to Semicomplete Digraphs. *Discrete Applied Mathematics*, 154(6):890–897, 2006.
doi:10.1016/j.dam.2005.11.006 → *p. 20*

[GRY08a] G. Gutin, A. Rafiey, and A. Yeo. Minimum Cost Homomorphism Dichotomy for Oriented Cycles. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM'08)*, volume 5034 of *Lecture Notes in Computer Science*, pages 224–234. Springer, 2008.
doi:10.1007/978-3-540-68880-8_22 → *p. 20*

[GRY08b] G. Gutin, A. Rafiey, and A. Yeo. Minimum Cost Homomorphisms to Semicomplete Multipartite Digraphs. *Discrete Applied Mathematics*, 156(12):2429–2435, 2008.
doi:10.1016/j.dam.2007.09.023 → *p. 20*

[GRYT06] G. Gutin, A. Rafiey, A. Yeo, and M. Tso. Level of Repair Analysis and Minimum Cost Homomorphisms of Graphs. *Discrete Applied Mathematics*, 154(6):881–889, 2006.
doi:10.1016/j.dam.2005.06.012 → *pp. 9, 20*

[GS88] G. Gallo and B. Simeone. On the supermodular knapsack problem. *Mathematical Programming*, 45(1-3):295–309, 1988.
doi:10.1007/BF01589108 → *pp. 91, 111*

[GS08] G. Gottlob and S. Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems. *The Computer Journal*, 51(3):303–325, 2008.
doi:10.1093/comjnl/bxm056 → *p. 12*

[GT88] A. Goldberg and R. Tarjan. A New Approach to the Maximum Flow Problem. *Journal of the ACM*, 35(4):921–940, 1988.
doi:10.1145/48014.61051 → *pp. 15, 16, 88*

[GW95] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.
doi:10.1145/227683.227684 → *p. 30*

[Ham65] P. L. Hammer. Some network flow problems solved with pseudo-Boolean programming. *Operations Research*, 13(3):388–399, 1965.
doi:10.1287/opre.13.3.388 → *pp. 83, 87*

[HN90] P. Hell and J. Nešetřil. On the Complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990.
doi:10.1016/0095-8956(90)90132-J → *pp. 17, 19*

[HN04] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004. → *pp. 4, 5, 17, 18, 19, 55*

[HN08] P. Hell and J. Nešetřil. Colouring, constraint satisfaction, and complexity. *Computer Science Review*, 2(3):143–163, 2008.
doi:10.1016/j.cosrev.2008.10.003 → *pp. 17, 26*

[HS86] P. Hansen and B. Simeone. Unimodular functions. *Discrete Applied Mathematics*, 14(3):269–281, 1986.
doi:10.1016/0166-218X(86)90031-4 → *p. 115*

[IFF01] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM*, 48(4):761–777, 2001.
doi:10.1145/502090.502096 → *p. 30*

[IMM+07] P. M. Idziak, P. Marković, R. McKenzie, M. Valeriote, and R. Willard. Tractability and learnability arising from algebras with few subpowers. In *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science (LICS'07)*, pages 213–224. 2007.
doi:10.1109/LICS.2007.50 → *p. 17*

[IO09] S. Iwata and J. B. Orlin. A Simple Combinatorial Algorithm for Submodular Function Minimization. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*, pages 1230–1237. 2009.
doi:10.1145/1496770.1496903 → *pp. 30, 116*

[Iwa02] S. Iwata. A fully combinatorial algorithm for submodular function minimization. *Journal of Combinatorial Theory, Series B*, 84(2):203–212, 2002.
doi:10.1006/jctb.2001.2072 → *p. 30*

[Iwa03] S. Iwata. A faster scaling algorithm for minimizing submodular functions. *SIAM Journal on Computing*, 32(4):833–840, 2003.
doi:10.1137/S0097539701397813 → *p. 30*

[Iwa08] S. Iwata. Submodular Function Minimization. *Mathematical Programming*, 112(1):45–64, 2008.
doi:10.1007/s10107-006-0084-2 → *p. 30*

[JC95] P. G. Jeavons and M. C. Cooper. Tractable Constraints on Ordered Domains. *Artificial Intelligence*, 79(2):327–339, 1995.
doi:10.1016/0004-3702(95)00107-7 → *pp. 54, 55*

[JCC98] P. Jeavons, D. Cohen, and M. C. Cooper. Constraints, Consistency and Closure. *Artificial Intelligence*, 101(1–2):251–265, 1998.
doi:10.1016/S0004-3702(98)00022-8 → *pp. 17, 85*

[JCG96] P. Jeavons, D. Cohen, and M. Gyssens. A Test for Tractability. In *Proceedings of the 2nd International Conference on Constraint Programming (CP'96), 1996*, volume 1118 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 1996.
doi:10.1007/3-540-61551-2 → *p. 34*

[JCG97] P. Jeavons, D. Cohen, and M. Gyssens. Closure Properties of Constraints. *Journal of the ACM*, 44(4):527–548, 1997.
doi:10.1145/263867.263489 → *pp. 5, 26, 35, 50*

[JCG99] P. Jeavons, D. Cohen, and M. Gyssens. How to Determine the Expressive Power of Constraints. *Constraints*, 4(2):113–131, 1999.
doi:10.1023/A:1009890709297 → *p. 34*

[Jea98] P. Jeavons. On the Algebraic Structure of Combinatorial Problems. *Theoretical Computer Science*, 200(1-2):185–204, 1998.
doi:10.1016/S0304-3975(97)00230-2 → *pp. 5, 26, 34*

[Jea09] P. G. Jeavons. Presenting Constraints. In *Proceedings of the 18th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'09)*, volume 5607 of *Lecture Notes in Artificial Intelligence*, pages 1–15. Springer, 2009.
doi:10.1007/978-3-642-02716-1_1 → *p. 9*

[Jég93] P. Jégou. Decomposition of Domains Based on the Micro-Structure of Finite Constraint-Satisfaction Problems. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)*, pages 731–736. 1993.
http://www.aaai.org/Library/AAAI/1993/aaai93-109.php → *p. 64*

[JK07] P. Jonsson and A. Krokhin. Maximum H-colourable subdigraphs and constraint optimization with arbitrary weights. *Journal of Computer and System Sciences*, 73(5):691–702, 2007.
doi:10.1016/j.jcss.2007.02.001 → *p. 18*

[JKK06] P. Jonsson, M. Klasson, and A. Krokhin. The Approximability of Three-valued MAX CSP. *SIAM Journal on Computing*, 35(6):1329–1349, 2006.
doi:10.1137/S009753970444644X → *pp. 18, 31*

[JKN08] P. Jonsson, F. Kuivinen, and G. Nordh. MAX ONES Generalized to Larger Domains. *SIAM Journal on Computing*, 38(1):329–365, 2008.
doi:10.1137/060669231 → *p. 18*

[JN08] P. Jonsson and G. Nordh. Introduction to the MAXIMUM SOLUTION Problem. In *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 255–282. Springer, 2008.
doi:10.1007/978-3-540-92800-3_10 → *p. 20*

[JNT07] P. Jonsson, G. Nordh, and J. Thapper. The maximum solution problem on graphs. In *Proceedings of the 32nd International Symposium on Mathematical Foundations of Computer Science (MFCS'07)*, volume 4708 of *Lecture Notes in Computer Science*, pages 228–239. Springer, 2007.
doi:10.1007/978-3-540-74456-6_22 → *p. 20*

[Jon00] P. Jonsson. Boolean constraint satisfaction: complexity results for optimization problems with arbitrary weights. *Theoretical Computer Science*, 244(1-2):189–203, 2000.
doi:10.1016/S0304-3975(98)00343-0 → *p. 18*

[KJJ03] A. Krokhin, P. Jeavons, and P. Jonsson. Reasoning about Temporal Relations: The Tractable Subalgebras of Allen's Interval Algebra. *Journal of the ACM*, 50(5):591–640, 2003.
doi:10.1145/876638.876639 → *p. 32*

[KKT07] P. Kohli, M. P. Kumar, and P. H. S. Torr. P3 & Beyond: Solving Energies with Higher Order Cliques. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'07)*. IEEE Computer Society, 2007.
doi:10.1109/CVPR.2007.383204 → *p. 115*

[KL08] A. Krokhin and B. Larose. Maximizing Supermodular Functions on Product Lattices, with Application to Maximum Constraint Satisfaction. *SIAM Journal on Discrete Mathematics*, 22(1):312–328, 2008.
doi:10.1137/060669565 → *pp. 30, 90*

[KLT09] P. Kohli, L. Ladický, and P. Torr. Robust Higher Order Potentials for Enforcing Label Consistency. *International Journal of Computer Vision*, 82(3):302–324, 2009.
doi:10.1007/s11263-008-0202-0 → *p. 86*

[KN08] G. Kun and J. Nešetřil. Forbidden lifts (NP and CSP for combinatorialists). *European Journal of Combinatorics*, 29(4):930–945, 2008.
doi:10.1016/j.ejc.2007.11.027 → *p. 17*

[KS09] G. Kun and M. Szegedy. A New Line of Attack on the Dichotomy Conjectur. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC'09)*, pages 725–734. 2009.
doi:10.1145/1536414.1536512 → *p. 17*

[KSTW01] S. Khanna, M. Sudan, L. Trevisan, and D. Williamson. The approximability of constraint satisfaction problems. *SIAM J. on Computing*, 30(6):1863–1920,

2001.
doi:10.1137/S0097539799349948 → *p. 18*

[Kun] G. Kun. Constraints, MMSNP and expander structures. *Combinatorica*. Submitted for publication. → *p. 17*

[KV00] P. Kolaitis and M. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. *Journal of Computer and System Sciences*, 61(2):302–332, 2000.
doi:10.1006/jcss.2000.1713 → *pp. 12, 14, 17*

[KV07a] P. G. Kolaitis and M. Y. Vardi. A Logical Approach to Constraint Satisfaction. In *Finite Model Theory and Its Applications*, Texts in Theoretical Computer Science. An EATCS Series. Springer, 2007. → *pp. 5, 14, 17*

[KV07b] B. Korte and J. Vygen. *Combinatorial Optimization*, volume 21 of *Algorithms and Combinatorics*. Springer, 4th edition, 2007. → *p. 30*

[Lad75] R. E. Ladner. On the Structure of Polynomial Time Reducibility. *Journal of the ACM*, 22(1):155–171, 1975.
doi:10.1145/321864.321877 → *p. 17*

[Lau96] S. L. Lauritzen. *Graphical Models*. Oxford University Press, 1996. → *pp. 31, 114*

[LD00] J. Larrosa and R. Dechter. On the Dual Representation of non-Binary Semiring-based CSPs. In *Workshop on Soft Constraints – CP'00*. 2000.
http://www.math.unipd.it/~frossi/cp2000-soft/program.html → *pp. 80, 115*

[Lew78] H. R. Lewis. Renaming a Set of Clauses as a Horn Set. *Journal of the ACM*, 25(1):134–135, 1978.
doi:10.1145/322047.322059 → *p. 81*

[Lov83] L. Lovász. Submodular Functions and Convexity. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming – The State of the Art*, pages 235–257. Springer, Berlin, 1983. → *p. 30*

[LRHB06] X. Lan, S. Roth, D. P. Huttenlocher, and M. J. Black. Efficient Belief Propagation with Learned Higher-Order Markov Random Fields. In *Proceedings of the 9th European Conference on Computer Vision (ECCV'06), Part II*, volume 3952 of *Lecture Notes in Computer Science*, pages 269–282. Springer, 2006.
doi:10.1007/11744047_21 → *p. 115*

[LT09] B. Larose and P. Tesson. Universal Algebra and Hardness Results for Constraint Satisfaction Problems. *Theoretical Computer Science*, 410(18):1629–1647, 2009.
doi:10.1016/j.tcs.2008.12.048 → *p. 5*

[LZ07] B. Larose and L. Zádori. Bounded width problems and algebras. *Algebra Universalis*, 56(3-4):439–466, 2007.
doi:10.1007/s00012-007-2012-6 → *p. 17*

[Mar04] D. Marx. *Graph Coloring with Local and Global Constraints*. Ph.D. thesis, Department of Computer Science and Information Theory, Budapest University of Technology and Economics, 2004.

[Mar07] D. Marx. Can you beat treewidth? In *Proocedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 169–179. IEEE Computer Society, 2007.
doi:10.1109/FOCS.2007.27 → *p. 14*

[Mar09a] D. Marx. Approximating fractional hypertree width. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*, pages 902–911. 2009.
doi:10.1145/1496770.1496868 → *p. 14*

[Mar09b] D. Marx. Tractable structures for constraint satisfaction with truth tables. In *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS'09)*, pages 649–660. 2009.
http://drops.dagstuhl.de/opus/volltexte/2009/1807/ → *p. 14*

[Mat70] Y. V. Matiyasevič. Enumerable sets are Diophantine. 191(2):279–282, 1970. In Russian. English Translation in: *Soviet Mathematical Doklady* 11, 354–357, 1970. → *p. 4*

[MF93] A. Mackworth and E. Freuder. The Complexity of Constraint Satisfaction Revisited. *Artificial Intelligence*, 59(1-2):57–62, 1993.
doi:10.1016/0004-3702(93)90170-G → *p. 13*

[MM08] M. Maróti and R. McKenzie. Existence theorems for weakly symmetric operations. *Algebra Universalis*, 59(3-4):463–489, 2008.
doi:10.1007/s00012-008-2122-9 → *p. 26*

[Mon74] U. Montanari. Networks of Constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.
doi:10.1016/0020-0255(74)90008-5 → *p. 9*

[MRTT53] T. Motzkin, H. Raiffa, G. Thompson, and R. Thrall. The double description method. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 2, pages 51–73. Princeton University Press, 1953. → *p. 109*

[Nar97] H. Narayanan. *Submodular Functions and Electrical Networks*. North-Holland, Amsterdam, 1997. → *p. 30*

[NB95] B. Nebel and H.-J. Bürckert. Reasoning About Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra. *Journal of the ACM*, 42(1):43–66, 1995.
doi:10.1145/200836.200848 → *p. 32*

[NI92] H. Nagamochi and T. Ibaraki. Computing Edge-Connectivity in Multigraphs and Capacitated Graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.
doi:10.1137/0405004 → *p. 16*

[NSZ09] J. Nešetřil, M. H. Siggers, and L. Zádori. A combinatorial constraint satisfaction problem dichotomy classification conjecture. *European Journal of Combinatorics*, 2009.
doi:10.1016/j.ejc.2009.02.007 → *p. 17*

[NW88] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988. → *p. 30*

[NWF78] G. Nemhauser, L. Wolsey, and M. Fisher. An Analysis of Approximations for Maximizing Submodular Set Functions-I. *Mathematical Programming*, 14(1):265–294, 1978.
doi:10.1007/BF01588971 → *p. 86*

[Orl09] J. B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118(2):237–251, 2009.
doi:10.1007/s10107-007-0189-2 → *pp. 30, 83*

[PK79] R. Pöschel and L. Kalužnin. *Funktionen- und Relationenalgebren*. DVW, Berlin, 1979. → *p. 37*

[PL98] R. Paget and I. D. Longstaff. Texture synthesis via a noncausal nonparametric multiscale Markov random field. *IEEE Transactions on Image Processing*, 7(6):925–931, 1998.
doi:10.1109/83.679446 → *p. 115*

[Pos41] E. Post. *The two-valued iterative systems of mathematical logic*, volume 5 of *Annals of Mathematical Studies*. Princeton University Press, 1941. → *p. 37*

[PQ82] J.-C. Picard and M. Queyranne. A network flow solution to some nonlinear 0-1 programming programs, with applications to graph theory. *Networks*, 12(2):141–159, 1982.
doi:10.1002/net.3230120206 → *p. 95*

[PR75] J.-C. Picard and H. Ratliff. Minimum cuts and related problems. *Networks*, 5(4):357–370, 1975.
doi:10.1002/net.3230050405 → *p. 95*

[PY05] S. Promislow and V. Young. Supermodular Functions on Finite Lattices. *Order*, 22(4):389–413, 2005.
doi:10.1007/s11083-005-9026-5 → *pp. 86, 98, 99, 102, 109, 110*

[Que98] M. Queyranne. Minimising symmetric submodular functions. *Mathematical Programming*, 82(1-2):3–12, 1998.
doi:10.1007/BF01585863 → *p. 30*

[Rag08] P. Raghavendra. Optimal algorithms and inapproximability results for every CSP? In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC'08)*, pages 245–254. 2008.
doi:10.1145/1374376.1374414 → *p. 18*

[Rao] A. Rao. On the Theory of Computing. Unpublished essay. → *p. 1*

[RB05] S. Roth and M. J. Black. Fields of experts: A framework for learning image priors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pages 860–867. IEEE Computer Society, 2005. doi:10.1109/CVPR.2005.160 → *p. 115*

[Rei08] O. Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4), 2008. doi:10.1145/1391289.1391291 → *p. 23*

[Rhy70] J. Rhys. A selection problem of shared fixed costs and network flows. *Management Science*, 17(3):200–207, 1970. http://www.jstor.org/stable/2629089 → *pp. 83, 87, 89, 95, 102*

[RKAT08] S. Ramalingam, P. Kohli, K. Alahari, and P. Torr. Exact Inference in Multi-label CRFs with Higher Order Cliques. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'08)*. IEEE Computer Society, 2008. doi:10.1109/CVPR.2008.4587401 → *p. 86*

[RKFJ09] C. Rother, P. Kohli, W. Feng, and J. Jia. Minimizing Sparse Higher Order Energy Functions of Discrete Variables. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'09)*. IEEE Computer Society, 2009. → *p. 115*

[Ros75] I. Rosenberg. Reduction of bivalent maximization to the quadratic case. *Cahier du Centre dEtudes de Recherche Opérationnelle*, 17:71–74, 1975. → *p. 83*

[RvBW06] F. Rossi, P. van Beek, and T. Walsh, editors. *The Handbook of Constraint Programming*. Elsevier, 2006. → *pp. 2, 4, 6, 9, 10, 11, 32*

[SBK00] M. Studený, R. R. Bouckaert, and T. Kočka. Extreme supermodular set functions over five variables. Technical Report number 1997, Institute of Information Theory and Automation, Prague, January 2000. http://www.utia.cas.cz/user_data/studeny/f14.html → *p. 116*

[Sch78] T. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC'78)*, pages 216–226. 1978. doi:10.1145/800133.804350 → *pp. 4, 16*

[Sch86] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Inc., 1986. → *pp. 28, 40, 42, 45, 46*

[Sch00] A. Schrijver. A Combinatorial Algorithm Minimizing Submodular Functions in Strongly Polynomial Time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000. doi:10.1006/jctb.2000.1989 → *p. 30*

[Sch03] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003. → *pp. 30, 85*

[Sch07] D. Schlesinger. Exact Solution of Permuted Submodular MinSum Problems. In *Proceedings of the 6th International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR'07)*, volume 4679 of *Lecture Notes in Computer Science*, pages 28–38. Springer, 2007. doi:10.1007/978-3-540-74198-5_3 → *p. 96*

[SdWC90] B. Simeone, D. de Werra, and M. Cochand. Recognition of a class of unimodular functions. *Discrete Applied Mathematics*, 29(2-3):243–250, 1990. doi:10.1016/0166-218X(90)90147-5 → *p. 96*

[SFV95] T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*. 1995. http://dli.iiit.ac.in/ijcai/IJCAI-95-VOL 1/ → *p. 9*

[SGG08] F. Scarcello, G. Gottlob, and G. Greco. Uniform Constraint Satisfaction Problems and Database Theory. In *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 156–195. Springer, 2008. doi:10.1007/978-3-540-92800-3_7 → *pp. 4, 5, 12*

[SJ08] A. Z. Salamon and P. G. Jeavons. Perfect Constraints Are Tractable. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming (CP'08)*, volume 5202 of *Lecture Notes in Computer Science*, pages 524–528. Springer, 2008. doi:10.1007/978-3-540-85958-1_35 → *p. 22*

[SS06] H. Schnoor and I. Schnoor. New Algebraic Tools for Constraint Satisfaction. In *Complexity of Constraints*, volume 06401 of *Dagstuhl Seminar Proceedings*. 2006. http://drops.dagstuhl.de/opus/volltexte/2006/805 → *p. 50*

[SW97] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997. doi:10.1145/263867.263872 → *p. 16*

[Top98] D. Topkis. *Supermodularity and Complementarity*. Princeton University Press, 1998. → *p. 30*

[Val79] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. doi:10.1016/0304-3975(79)90044-6 → *p. 20*

[Wer07] T. Werner. A Linear Programming Approach to Max-Sum Problem: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1165–1179, 2007. doi:10.1109/TPAMI.2007.1036 → *p. 114*

[WJ08] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008. doi:10.1561/2200000001 → *pp. 31, 114*

[Zal08] B. Zalesky. Efficient Determination of Gibbs Estimators with Submodular Energy Functions, February 2008. ArXiv:math/0304041v1.
http://arxiv.org/abs/math/0304041 → *p. 95*

[ZŽ09] B. Zanuttini and S. Živný. A note on some collapse results of valued constraints. *Information Processing Letters*, 109(11):534–538, 2009.
doi:10.1016/j.ipl.2009.01.018 → *p. 53*

[ŽCJ08] S. Živný, D. A. Cohen, and P. G. Jeavons. The Expressive Power of Binary Submodular Functions. Technical report, November 2008. ArXiv:0811.1885.
http://arxiv.org/abs/0811.1885

[ŽCJ09] S. Živný, D. A. Cohen, and P. G. Jeavons. The Expressive Power of Binary Submodular Functions. *Discrete Applied Mathematics*, 157(15):3347–3358, 2009.
doi:10.1016/j.dam.2009.07.001

[ŽJ08] S. Živný and P. G. Jeavons. Which submodular functions are expressible using binary submodular functions? Research Report CS-RR-08-08, Computing Laboratory, University of Oxford, Oxford, UK, June 2008.
http://web.comlab.ox.ac.uk/publications/publication85-abstract.html → *p. 114*

[ŽJ09a] S. Živný and P. G. Jeavons. Classes of Submodular Constraints Expressible by Graph Cuts. To appear in *Constraints*, 2009.
doi:10.1007/s10601-009-9078-z

[ŽJ09b] S. Živný and P. G. Jeavons. The complexity of valued constraint models. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP'09)*, volume 5732 of *Lecture Notes in Computer Science*, pages 833–841. Springer, 2009.
doi:10.1007/978-3-642-04244-7_64 → *pp. 16, 22, 51*

Index