Computing Science

WHICH SUBMODULAR FUNCTIONS ARE EXPRESSIBLE USING BINARY SUBMODULAR FUNCTIONS?

Stanislav Živný
Peter G. Jeavons

CS-RR-08-08

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Which submodular functions are expressible using binary submodular functions?[*]

Stanislav Živný[†]
Peter G. Jeavons

Abstract

Submodular functions occur in many combinatorial optimisation problems and a number of polynomial-time algorithms have been devised to minimise such functions. The time complexity of the fastest known general algorithm for submodular function minimisation (SFM) is $O(n^6 + n^5L)$, where n is the number of variables and L is the time required to evaluate the function.

However, many important special cases of SFM can be solved much more efficiently, and with much simpler algorithms. For example, the (s,t)-Min-Cut problem is a special case of SFM which can be solved in cubic time. Moreover, any submodular function which can be expressed as a sum of binary submodular functions can be minimised by computing a minimal cut in a suitable graph. It has been known for some time that all ternary submodular functions are expressible in this way, by introducing additional variables. We have recently identified, for each $k \geq 4$, a subclass of k-ary submodular functions which are also expressible in this way.

It was previously an open question whether all submodular functions could be expressed as a sum of binary submodular functions over a larger set of variables: in this paper we show that they cannot. Moreover, we characterise precisely which 4-ary submodular functions can be expressed in this way. This result can also be seen as characterising which pseudo-Boolean functions of degree 4 can be expressed as projections of quadratic submodular functions.

Our results provide a more efficient algorithm for certain discrete optimisation problems which can be formulated as valued constraint satisfaction problems (VCSP). We define a new maximal class of VCSP instances with submodular constraints which are expressible using binary submodular functions with a bounded number of additional variables. It follows that optimal solutions to such instances can be computed in $O((n+k)^3)$ time, where n is the number of variables and k is the number of higher-order (non-binary) constraints, by a straightforward reduction to the (s,t)-Min-Cut problem.

[*]Further copies of this Research Report may be obtained from the Librarian, Oxford University Computing Laboratory, Computing Science, Wolfson Building, Parks Road, Oxford OX1 3QD, England (Telephone: +44-1865-273837, Email: `library@comlab.ox.ac.uk`).

[†]Stanislav Živný gratefully acknowledges the support of EPSRC grant EP/F01161X/1.

1 Introduction

The CONSTRAINT SATISFACTION PROBLEM (CSP) is a general framework which can be used to model many different problems [12, 25, 30]. However, the CSP model considers only the feasibility of satisfying a collection of simultaneous requirements (so-called *hard constraints*).

Various extensions have been proposed to this model to allow it to deal with different kinds of optimisation criteria, or preferences, between different feasible solutions (so-called *soft constraints*). Two very general extended frameworks that have been proposed are the SCSP (semi-ring CSP) framework and the VCSP (valued CSP) framework [2]. The SCSP framework is slightly more general[1], but the VCSP framework is simpler, and yet sufficiently powerful to model a wide range of optimisation problems [2, 30, 31]. In particular, it generalises the classical CSP model, and includes many standard optimisation problems such as MIN-CUT, MAX-SAT, MAX-ONES SAT and MAX-CSP [9, 12], and also MIN-COST HOMOMORPHISM [19].

Informally, in the VALUED CONSTRAINT SATISFACTION PROBLEM (VCSP) framework, 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. 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 (COP) framework [30], which is widely used in practice.

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 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 $\mathcal{P}$. 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 [14, 16, 17]. In fact, in the case of bounded-arity CSPs, the question of identifying all structural restrictions which guarantee tractability has been resolved completely, see [18] for details.

Second, we can restrict the *forms* of the valued constraints which are allowed in the problem, that is, the set of possible cost functions. This gives rise to so-called *language restrictions*.

Several language restrictions which ensure tractability have been identified in the

[1]The main difference is that costs in VCSPs represent violation levels and have to be totally ordered, whereas costs in SCSPs represent preferences and might be ordered only partially.

literature, (see, e.g., [9]). One important and well-studied restriction on cost functions is *submodularity* [7]. The class of constraints with submodular cost functions is the only non-trivial tractable case in the dichotomy classification of the Boolean VCSP [9], and the only tractable case in the dichotomy classification of the Max-CSP problem for both 3-element sets [22] and arbitrary finite sets allowing constant constraints [13].

The notion of submodularity originally comes from combinatorial optimisation where submodular functions are defined on subsets of a given base set [20, 26, 34, 33, 15]. Note that the minimisation of submodular functions on sets is equivalent to the minimisation of submodular functions on distributive lattices. Krokhin and Larose have also studied the minimisation of submodular functions on non-distributive lattices [24].

The time complexity of the fastest known algorithm for the problem of Submodular Function Minimisation (SFM), which consists in minimising a function ψ defined on subsets of V, is $O(n^6 + n^5 L)$, where $n = |V|$ and L is the time to evaluate $\psi(S)$ for any $S \subseteq V$ [27]. However, there are several known special cases of SFM that can be solved more efficiently than the general case (see [3] for a survey).

Cohen et al. showed that VCSPs with submodular constraints over an arbitrary finite domain can be reduced to the SFM problem over a special family of sets known as a ring family [9]. This problem is equivalent to the general SFM problem [32], thus giving an algorithm of order $O(n^6 + n^5 L)$, where L is the look-up time (needed to evaluate an assignment to all variables), for any VCSP with submodular constraints. This tractability result has since been generalised to a wider class of valued constraints over arbitrary finite domains known as tournament-pair constraints [6]. An alternative approach to solving VCSP instances with bounded-arity submodular constraints, based on linear programming, can be found in [10].

For certain forms of submodular constraints a simpler and more efficient algorithm is possible. In particular, we shall define a notion of *expressibility* below, and show that any VCSP with submodular constraints which are expressible using *binary* submodular constraints (over a bounded number of additional variables) can be reduced to the (s,t)-Min-Cut problem. Such problems can be solved in $O((n+k)^3)$ time, where n is the number of variables and k is the number of higher-order (non-binary) constraints, by finding a minimum cut in a suitable graph.

Hence the main research problem we will look at in this paper is the following.

Problem: Which submodular constraints are expressible using binary submodular constraints?

It is known that all *ternary* submodular constraints are expressible using binary submodular constraints, with a single additional variable. Hence the class of all VCSP instances with submodular constraints of arity 3 or lower can be reduced to (s,t)-Min-Cut (see [1, 35] for details).

In addition, Živný and Jeavons [35] identified certain special subclasses of submodular constraints of all arities which are expressible using binary submodular constraints. The class of VCSP instances with submodular constraints from these subclasses can therefore also be solved efficiently by reducing to (s,t)-Min-Cut. Similar results were obtained independently by Zalesky [36].

These previous results naturally raised the question of whether *all* submodular constraints can be expressed using binary submodular constraints: in this paper we show that they cannot.

In particular, we carefully investigate 4-ary submodular constraints, and obtain a complete characterisation for this case.

Result: Characterisation of which 4-ary submodular constraints are expressible using binary submodular constraints.

The concept of submodularity not only plays an important role in theory, but is also very important in practice. For example, many of the problems that arise in computer vision can be expressed in terms of energy minimisation [23], which is easily expressible in the VCSP framework. Kolmogorov and Zabih identified classes of instances of arity at most three for which the energy minimisation problem can be solved efficiently by reducing to (s,t)-Min-Cut [23], and which are applicable to a wide variety of vision problems, including image restoration, stereo vision and motion tracking, image synthesis, image segmentation, multi-camera scene reconstruction and medical imaging. The so-called *regularity* condition, which specifies the efficiently solvable classes in [23], turned out to be just a rediscovery of the well-known submodularity condition studied before in different areas of computer science. We believe that our result on new classes of submodular constraints which are expressible using binary submodular constraints will also find application in computer vision.

The paper is organised as follows. In Section 2, we define the VCSP framework and the notion of expressibility, and describe certain key algebraic properties of valued constraints. Also, we define submodular cost functions and observe that finite-valued cost functions over a Boolean domain can be represented by polynomials, so the Boolean VCSP is equivalent to pseudo-Boolean function minimisation. In Section 3 we translate into this framework some recent results from [28] concerning the generators of the cone of 4-ary submodular functions. In Section 4, we present our main result characterising the expressibility of such functions and use it to identify a new tractable subclass of the VCSP. Finally, in Section 5, we summarise our work.

2 Preliminaries

In this section we introduce the basic definitions and the main tools used throughout the paper.

2.1 VCSP

We first define the very general framework for discrete optimisation problems known as the valued constraint satisfaction problem (VCSP). In the original definition of this problem, given in [31], costs were allowed to lie in any positive totally ordered monoid called a *valuation structure*. For our purposes, it is sufficient to consider costs which lie in the set $\overline{\mathbb{Q}}_+$ consisting of all non-negative rational numbers together with infinity[2]. (Actually, as any cost function can be scaled up, without loss of generality, we can simply assume that all costs are non-negative integers or infinite).

Given a fixed set D, a function from D^k to $\overline{\mathbb{Q}}_+$ will be called a *cost function*. If the range of ϕ lies entirely within $\mathbb{Q}_+$, the set of non-negative rationals, 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.

Note that with any *relation* R over D we can associate a crisp cost function ϕ_R on D which maps tuples in R to 0 and tuples not in R to ∞. On the other hand, with any m-ary cost function ϕ we can associate a relation R_ϕ defined as $\langle x_1, \ldots, x_m\rangle \in R_\phi \Leftrightarrow \phi(x_1, \ldots, x_m) < \infty$. Therefore, crisp cost functions correspond precisely to relations, so we shall use these terms interchangeably.

Definition 2.1. *An instance $\mathcal{P}$ of* 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 $c = \langle \sigma, \phi\rangle$, where σ is a tuple of variables of length $|\sigma|$, called the* scope *of c, and $\phi : D^{|\sigma|} \to \overline{\mathbb{Q}}_+$ is a cost function. An* assignment *for the instance $\mathcal{P}$ is a mapping s from V to D. The* cost *of an assignment s is defined as follows:*

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

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

Any set Γ of cost functions is called a *valued constraint language*. The class VCSP(Γ) is defined to be the class of all VCSP instances where the cost functions of all valued constraints lie in Γ.

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

Definition 2.2. *For any* VCSP *instance $\mathcal{I} = \langle V, D, \mathcal{C}\rangle$, and any list of variables of $\mathcal{I}$, $l = \langle v_1, \ldots, v_m\rangle$, the* projection *of $\mathcal{I}$ onto l, denoted $\pi_l(\mathcal{I})$, is the m-ary cost function defined as follows:*

$$\pi_l(\mathcal{I})(x_1, \ldots, x_m) \; = \; \min_{\{s:V\to D \mid \langle s(v_1), \ldots, s(v_m)\rangle = \langle x_1, \ldots, x_m\rangle\}} Cost_{\mathcal{I}}(s).$$

We say that a cost function ϕ is expressible *over a valued constraint language Γ if there exists an instance $\mathcal{I} \in$ VCSP(Γ) and a list l of variables of $\mathcal{I}$ such that $\pi_l(\mathcal{I}) = \phi$.*

[2] See [11] for a discussion of why limiting ourselves to the $\overline{\mathbb{Q}}_+$ valuation structure is not a severe restriction.

We call the pair $\langle \mathcal{I}, l\rangle$ a gadget *for expressing ϕ over Γ. Variables from $V \setminus l$ are called* extra *or* hidden *variables.*

Note that in the special case of relations (crisp cost functions) this notion of expressibility corresponds to the standard notion of expressibility using conjunction and existential quantification (*primitive positive formulas*) [4].

We denote by $\langle\Gamma\rangle$ the *expressive power* of Γ, which is the set of all cost functions expressible over Γ up to additive and multiplicative constants.

2.2 Algebraic properties of cost functions

In this section we define certain algebraic properties of cost functions and show that these properties characterise the expressive power of any valued constraint language.

The i-th component of a tuple $\mathbf{t}$ 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 $\mathbf{t}_1, \ldots, \mathbf{t}_k \in D^m$, define $f(\mathbf{t}_1, \ldots, \mathbf{t}_k) \in D^m$ to be the tuple $\langle f(t_1[1], \ldots, t_k[1]), \ldots, f(t_1[m], \ldots, t_k[m])\rangle$.

It was shown in [5] that the expressive power of a finite-valued constraint language is fully captured by the following algebraic properties (see Figure 1 for an illustration of Definition 2.3).

Definition 2.3 ([5])**.** *A k-ary* weighted function *$\mathcal{F}$ on a set D is a set of the form $\{\langle r_1, f_1\rangle, \ldots, \langle r_n, f_n\rangle\}$ where each r_i is a non-negative rational number such that $\sum_{i=1}^{n} r_i = k$ and each f_i is a distinct function from D^k to D.*

For any m-ary cost function ϕ, we say that a k-ary weighted function $\mathcal{F}$ is a k-ary fractional polymorphism *of ϕ if, for all $\mathbf{t}_1, \ldots, \mathbf{t}_k \in D^m$,*

$$\sum_{i=1}^{k} \phi(\mathbf{t}_i) \;\geq\; \sum_{i=1}^{n} r_i \phi(f_i(\mathbf{t}_1, \ldots, \mathbf{t}_k)).$$

A k-ary fractional polymorphism where each r_i is a natural number is called a k-ary multimorphism[3]*.*

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

Theorem 2.4 ([5])**.** *If Γ is a finite-valued constraint language which is closed under scaling*[4]*, then*

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

[3]In this case $\mathcal{F}$ can be written as $\langle f_1, \ldots, f_k\rangle$ (where the f_is are not necessarily distinct) and therefore $\mathcal{F}$ can be seen as a mapping from D^k to D^k.

[4]That is, $\phi \in \Gamma$ and $c \in \mathbb{Q}_+$ implies $c\phi \in \Gamma$.

$$\begin{array}{ccccccc}
t_1 & t_1[1] & t_1[2] & \dots & t_1[k] & & \phi(t_1) \\
t_2 & t_2[1] & t_2[2] & \dots & t_2[k] & \xrightarrow{\phi} & \phi(t_2) \\
\vdots & & & \vdots & & & \vdots \\
t_k & t_k[1] & t_k[2] & \dots & t_k[k] & & \phi(t_k) \\
\hline
t'_1 = f_1(t_1,\dots,t_k) & t'_1[1] & t'_1[2] & \dots & t'_1[k] & & \phi(t'_1) \\
t'_2 = f_2(t_1,\dots,t_k) & t'_2[1] & t'_2[2] & \dots & t'_2[k] & \xrightarrow{\phi} & \phi(t'_2) \\
\vdots & & & \vdots & & & \vdots \\
t'_n = f_n(t_1,\dots,t_k) & t'_n[1] & t'_n[2] & \dots & t'_n[k] & & \phi(t'_n)
\end{array}
\quad
\begin{array}{c}
\left.\right\} \sum_{i=1}^{k} \phi(t_i) \\
\geq \\
\left.\right\} \sum_{i=1}^{n} r_i \phi(t'_i)
\end{array}$$

Figure 1: Definition of a fractional polymorphism $\mathcal{F} = \{\langle r_1, f_1\rangle, \dots, \langle r_n, f_n\rangle\}$.

Hence, for all finite-valued constraint languages closed under scaling, a cost function ϕ is expressible over Γ if and only if it has all the fractional polymorphisms of Γ (including all multimorphisms).

2.3 Submodular functions

A function $\psi : 2^V \to \mathbb{Q}$ defined on subsets of a set V is called a *submodular function* [26] if, for all subsets S and T of V, $\psi(S \cap T) + \psi(S \cup T) \leq \psi(S) + \psi(T)$. The problem of Submodular Function Minimisation (SFM) consists in finding a subset S of V for which the value of $\psi(S)$ is minimal.

For any lattice-ordered set D, a cost function $\phi : D^k \to \overline{\mathbb{Q}}_+$ is called *submodular* if for every $u, v \in D^k$, $\phi(\min(u,v)) + \phi(\max(u,v)) \leq \phi(u) + \phi(v)$ where both min and max are applied coordinate-wise on tuples u and v. In other words, ϕ is submodular if $\langle \textsc{Min}, \textsc{Max}\rangle \in \mathsf{Mul}(\{\phi\})$. Note that expressibility preserves submodularity: if every $\phi \in \Gamma$ is submodular, and $\phi' \in \langle\Gamma\rangle$, then ϕ' is also submodular. (This is a consequence of Theorem 2.4: expressibility preserves fractional polymorphisms.)

Next we show that when considering which cost functions are expressible over binary submodular cost functions, we can restrict our attention to Boolean finite-valued cost functions without any loss of generality. This important simplification seems not to be widely known in the computer vision community, so we include it here for completeness.

First, it is easy to see that any VCSP instance with n variables over a non-Boolean domain $\{0, \dots, d-1\}$ of size d can be encoded as a VCSP instance with $O(nd)$ variables over the Boolean domain $\{0, 1\}$. One such encoding is the following: $en(i) = 0^{d-i-1}1^i$. We replace each variable with d 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 $en(\max(a,b)) = \max(en(a), en(b))$ and $en(\min(a,b)) = \min(en(a), en(b))$, so this encoding preserves submodularity.

Next, using results from [9] and [33], it can be shown that any submodular cost function ϕ can be expressed as the sum of a finite-valued submodular cost function ϕ_{fin}, and a submodular relation ϕ_{crisp}, that is, $\phi = \phi_{fin} + \phi_{crisp}$.

Moreover, it is known that all submodular *relations* are binary decomposable [21], and hence expressible using only binary submodular relations. Therefore, when considering which cost functions are expressible over binary submodular cost functions, we can restrict our attention to *Boolean finite-valued* cost functions without any loss of generality.

Therefore, in this paper we focus on finite-valued problems over Boolean domains. We denote by $\Gamma_{\mathsf{sub,k}}$ the set of all finite-valued submodular cost functions of arity at most k on the Boolean domain $D = \{0, 1\}$, and we set $\Gamma_{\mathsf{sub}} = \bigcup_k \Gamma_{\mathsf{sub,k}}$. We will show below that VCSP($\Gamma_{\mathsf{sub,2}}$) can be solved in cubic time, and hence we will be concerned with what other cost functions are expressible over $\Gamma_{\mathsf{sub,2}}$, and so can also be solved efficiently.

2.4 Polynomials and pseudo-Boolean functions

A cost function of arity k can be represented as a table of values of size D^k. Alternatively, a (finite-valued) cost function $\phi : D^k \to \mathbb{Q}_+$ on a Boolean domain $D = \{0, 1\}$ can be uniquely represented as a *polynomial* in k (Boolean) variables with coefficients from $\mathbb{Q}$ (such functions are sometimes called *pseudo-Boolean functions* [3]). Hence, in what follows, we will often represent a finite-valued cost function on a Boolean domain by a polynomial.

If Γ is a set of cost functions on a Boolean domain, with arity at most k, then any instance of VCSP(Γ) 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 that in any Boolean domain $x^2 = x$, so p has at most 2^n terms which correspond to subsets of variables.

For polynomials over Boolean variables there is a standard way to define *derivatives* of each order (see [3]). For example, the second order derivative of a polynomial p, with respect to the first two indices, denoted $\delta_{1,2}(\vec{x})$, is defined as $p(1, 1, \vec{x}) - p(1, 0, \vec{x}) - p(0, 1, \vec{x}) + p(0, 0, \vec{x})$. Analogously for all other pairs of indices.

Proposition 2.5 ([3]). *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 $\delta_{i,j}(\vec{x})$ are non-positive for all $1 \leq i < j \leq n$ and all $\vec{x} \in D^{n-2}$.*

Corollary 2.6. *A* quadratic *polynomial $a_0 + \sum_{i=1}^{n} a_i x_i + \sum_{1 \leq i < j \leq n} a_{ij} x_i x_j$ over Boolean variables $x_1, \ldots, x_k$, represents a submodular cost function if and only if $a_{ij} \leq 0$ for every $1 \leq i < j \leq n$.*

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. 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 (see [3] for details).

2.5 VCSPs with submodular constraints

Using known results for the SFM problem, Cohen et al. established the tractability of VCSP for submodular constraints over arbitrary finite domains.

Theorem 2.7 ([9]). VCSP(Γ_{sub}) *is solvable in* $O(n^6 + n^5 L)$ *time, where* n *is the number of variables and* L *is the look-up time (needed to evaluate an assignment to all variables).*

Now we show that the constraint language Γ_{cut}, consisting of certain simple binary and unary cost functions over a Boolean domain, captures precisely the (s,t)-Min-Cut problem, and hence has cubic time complexity. We will then show that Γ_{cut} can express any binary submodular cost function over a Boolean domain, that is, $\Gamma_{\mathsf{sub},2} \subseteq \langle\Gamma_{\mathsf{cut}}\rangle$. Hence we show that any instance of VCSP($\Gamma_{\mathsf{sub},2}$) can be solved in cubic time.

For any $w \in \overline{\mathbb{Q}}_+$, we define the binary cost function χ^w as follows:

$$\chi^w(x,y) \;=\; \begin{cases} w & \text{if } (x,y) = (0,1), \\ 0 & \text{otherwise.} \end{cases}$$

For any $d \in D$ and $c \in \overline{\mathbb{Q}}_+$, we define the unary cost function μ_d^c as follows:

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

It is straightforward to check that all χ^w and μ_d^c are submodular.

We define the constraint language Γ_{cut} to be the set of all cost functions χ^w and μ_d^c over a Boolean domain, for $w, c \in \overline{\mathbb{Q}}_+$ and $d \in \{0,1\}$.

Theorem 2.8 ([35]). *The problems* (s,t)-Min-Cut *and* VCSP(Γ_{cut}) *are linear-time equivalent.*

Corollary 2.9 ([35]). VCSP(Γ_{cut}) *is solvable in* $O(n^3)$ *time where* n *is the number of variables.*

Using a standard reduction (see, for example, [3]), it was shown in [35] that all binary submodular cost functions over a Boolean domain can be expressed over Γ_{cut}.

Theorem 2.10 ([35]). $\Gamma_{\mathsf{sub},2} \subseteq \langle\Gamma_{\mathsf{cut}}\rangle$.

As only a fixed number of extra variables is needed, we obtain the following:

Corollary 2.11 ([35]). VCSP($\Gamma_{\mathsf{sub},2}$) *is solvable in* $O(n^3)$ *time where* n *is the number of variables.*

Furthermore, all ternary submodular constraints are known to be expressible over $\Gamma_{\mathsf{sub},2}$ with one extra variable per ternary constraint [1].

The next result identifies several further classes of submodular constraints which are expressible using binary submodular constraints, and therefore solvable efficiently via reduction to the (s,t)-Min-Cut problem on weighted directed graphs.

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. It is easy to check that these cost functions, sometimes called *negative-positive*, are submodular. Set $\Gamma_{\mathsf{neg}} = \bigcup_k \Gamma_{\mathsf{neg,k}}$. The minimisation of cost functions chosen from Γ_{neg} using min-cuts was first studied in [29].

Define $\Gamma_{\{0,1\},\mathsf{k}}$ to be the set of all $\{0,1\}$-valued submodular cost functions over a Boolean domain, of arity at most k, and set $\Gamma_{\{0,1\}} = \cup_k \Gamma_{\{0,1\},\mathsf{k}}$. The minimisation of submodular cost functions from $\Gamma_{\{0,1\}}$ was studied in [12], where they were called *2-monotone* functions. The equivalence of 2-monotone and submodular cost functions, and a generalisation of 2-monotone cost functions to non-Boolean domains, was shown in [8].

Define $\Gamma_{\mathsf{new,k}}$ to be the set of all k-ary submodular cost functions over a Boolean domain whose corresponding polynomials satisfy, for every $1 \leq i < j \leq k$,

$$a_{ij} + \sum_{s=1}^{k-2} \sum_{\{i,j,i_1,\ldots,i_s\} \in C^+_{s+2}} a_{i,j,i_1,\ldots,i_s} \quad \leq \quad 0.$$

In other words, for any $1 \leq i < j \leq k$, 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. Set $\Gamma_{\mathsf{new}} = \bigcup_k \Gamma_{\mathsf{new,k}}$.

Theorem 2.12 ([35]). *$\Gamma_{\mathsf{sub,3}}$, Γ_{neg}, $\Gamma_{\{0,1\}}$, $\Gamma_{\mathsf{new}} \subseteq \langle \Gamma_{\mathsf{sub,2}} \rangle$;*

Consequently, VCSP instances over any of the above-mentioned languages can be solved by reducing to (s,t)-MIN-CUT.

Corollary 2.13 ([35]). *VCSP($\Gamma_{\mathsf{sub,3}}$) is solvable in $O((n+k)^3)$ time where n is the number of variables and k is the number of ternary constraints.*

Corollary 2.14 ([35]). *For any fixed k, VCSP($\Gamma_{\mathsf{neg,k}}$), VCSP($\Gamma_{\{0,1\},\mathsf{k}}$) and VCSP($\Gamma_{\mathsf{new,k}}$) are solvable in $O((n+k)^3)$ time where n is the number of variables*

3 Generators of 4-ary submodular cost functions

One way to answer questions about the expressibility of a set of cost functions Γ is to find a finite set of *generators* for Γ, in other words, a set of cost functions $\{\xi_1, \ldots, \xi_m\}$ such that every $\phi \in \Gamma$ can be written as $\sum_{i=1}^{m} \alpha_i \xi_i$ for $\alpha_i \geq 0$, $i = 1, 2, \ldots, m$.

In the following, we translate into the VCSP framework a recent result from [28] which identifies a finite set of generators for $\Gamma_{\mathsf{sub,4}}$. (Note that [28] actually studies supermodular functions, but as f is supermodular if and only if $-f$ is submodular, the results translate easily.)

In order to simplify the presentation, from this point onwards we allow cost functions to take (finite) negative values. Obviously, we could add a constant to them and get functions that take non-negative values only, but this would result in a more cumbersome presentation.

Recall that L is a lattice if L is a partially ordered set in which every pair of elements (a,b) has a unique supremum (the elements' least upper bound, called the *join*, denoted $a \vee b$) and an infimum (greatest lower bound, called the *meet*, denoted $a \wedge b$).

Definition 3.1 ([28]). *Let L be a lattice. An* upper fan *F in L is a subset of pairwise incomparable elements $(a_1, \ldots, a_m)$, for which each pair of distinct elements (a_i, a_j) has the same least upper bound. We call the index m the* length of the fan. *A* lower fan *G in L is a subset of incomparable elements $(a_1, \ldots, a_m)$, for which each pair of distinct elements (a_i, a_j) has the same greatest lower bound.*

Definition 3.2. *Let L be a lattice. We define the following functions on L:*

- *For any $x \in L$, the* upper function *u_x takes the value -1 on all points of L greater than or equal to x and the value 0 elsewhere;*
- *For any $x \in L$, the* lower function *l_x takes the value -1 on all points of L less than or equal to x and the value 0 elsewhere;*
- *For any upper fan F, where b is the common least upper bound of any pair of elements, the function u_F takes the value -1 on all elements x such that for some i, $a_i \leq x$, but $b \not\leq x$; the value -2 on all x such that $b \leq x$; and the value 0 elsewhere;*
- *For any lower fan G, where c is the common greatest lower bound of any pair of elements, the function l_G takes the value -1 on all elements x such that for some i, $x \leq a_i$, but $x \not\leq c$; the value -2 on all x such that $x \leq c$; and the value 0 elsewhere.*

Proposition 3.3 ([28]). *The functions u_x, l_x, u_F and l_G are submodular.*

Definition 3.4. *We say that $a \in L$ is* quasi-indecomposable *if $x \vee y = a$ implies that $x \wedge y = \bot$ (where $\bot$ is the minimum element of L) and $x \wedge y = a$ implies that $x \vee y = \top$ (where $\top$ is the maximum element of L). For a quasi-indecomposable a, we define qin_a to be the function that takes the value 1 at a, the value -1 at $\bot$ and at $\top$, and the value 0 elsewhere.*

Proposition 3.5 ([28]). *The function qin_a is submodular.*

Let Z_2^k be the lattice of all subsets of $\{1, 2, \ldots, k\}$. Note that cost functions defined on Z_2^k correspond precisely to k-ary Boolean cost functions.

Definition 3.6. *Define the following classes of submodular cost functions on Z_2^4, where i, j, k, l are distinct elements of $\{1, 2, 3, 4\}$:*

0. *$G_0 = \{f_{-1}, f_{+1}\} \cup \{f_i \mid 1 \leq i \leq 4\}$ where f_{+1} and f_{-1} are the constant cost functions which return 1 and -1, respectively, and f_i is the cost function which returns 1 if its i-th argument is 1 and 0 otherwise.*
1. *$G_1 = \{u_x \mid x = \{i, j\}\}$;*
2. *$G_2 = \{u_x \mid x = \{i, j, k\}\}$;*
3. *$G_3 = \{u_x \mid x = \{1, 2, 3, 4\}\}$;*

4. $G_4 = \{l_\emptyset\}$;

5. $G_5 = \{l_B \mid B = \{\{i\},\{j\},\{k\}\}\}$;

6. $G_6 = \{u_F \mid F = \{\{i,j\},\{i,k\},\{j,k\}\}\}$;

7. $G_7 = \{u_F \mid F = \{\{i,j,k\},\{i,j,l\},\{j,k,l\}\}\}$;

8. $G_8 = \{u_F \mid F = \{\{1,2,3\},\{1,2,4\},\{1,3,4\},\{2,3,4\}\}\}$;

9. $G_9 = \{u_F \mid F = \{\{i,j,k\},\{i,j,l\},\{k,l\}\}\}$;

10. $G_{10} = \{qin_a \mid a = \{i,j\}\}$.

Examples of cost functions from classes G_1 to G_{10} can be found in Figures 6 to 15 in Appendix B.

Given a set of functions $\mathcal{S}$, we denote by $\mathcal{L}(\mathcal{S})$ the non-negative span of $\mathcal{S}$, that is,

$$\mathcal{L}(\mathcal{S}) \;=\; \bigcup_{m\geq 0}\{\sum_{i=1}^{m}\alpha_i x_i \mid x_i \in \mathcal{S} \wedge \alpha_i \geq 0\}.$$

Theorem 3.7 (adapted from Theorem 5.2 of [28])**.** $\Gamma_{\mathsf{sub},4} = \mathcal{L}(\bigcup_{i=0}^{10} G_i)$

4 The main result

The next theorem characterises precisely which 4-ary submodular cost functions are expressible using only binary submodular cost functions.

Theorem 4.1 (Main result)**.** $\Gamma_{\mathsf{sub},4} \cap \langle\Gamma_{\mathsf{sub},2}\rangle = \mathcal{L}(\bigcup_{i=0}^{9} G_i)$

Corollary 4.2. VCSP$(\mathcal{L}(\bigcup_{i=0}^{9} G_i))$*, is solvable in* $O((n+k)^3)$ *time where* n *is the number of variables and* k *is the number of non-binary constraints.*

In order to prove Theorem 4.1, we have to show that any $f \in \bigcup_{i=0}^{9} G_i$ is expressible over $\Gamma_{\mathsf{sub},2}$, and also that no $f \in G_{10}$ is expressible over $\Gamma_{\mathsf{sub},2}$.

Figure 2 shows the number of cost functions in each class G_i, and an example of a cost function[5] from each G_i (the other cost functions from each G_i can be obtained easily by permuting variables.)

In three cases (G_4, G_5, G_{10}), instead of listing a member of G_i we have listed an alternative cost function obtained by adding multiples of cost functions from G_0. For example, the cost function $l'_\emptyset$, is obtained from the function $l_\emptyset \in G_4$ by subtracting constant and linear functions. The function $l'_\emptyset$ is represented in Figure 16 in Appendix B. Similarly for cost functions l'_{123} (represented in Figure 17), and qin'_{12} (represented in Figure 18). Using these alternative cost functions does not change the non-negative span of the generators, but simplifies the polynomial representations.

[5]To simplify notation in Figure 2, we omit curly brackets, which are normally used to denote sets. For instance, we write 123 instead of $\{1,2,3\}$.

Table 2 also shows the corresponding polynomial representation for each of these functions, and in most cases a corresponding gadget which can be used to express each of these functions over $\Gamma_{\mathsf{sub},2}$. Note that no gadget is needed for cost functions from G_0 or G_1 as $G_0, G_1 \subset \Gamma_{\mathsf{sub},2}$. Furthermore, note that no gadget over $\Gamma_{\mathsf{sub},2}$ is listed in Figure 2 for the example cost function from G_{10}. We will prove later that such a gadget does not exist.

G_1	6	u_{12}	$-x_1x_2$	
G_2	4	u_{123}	$-x_1x_2x_3$	$\min_y(y(2-x_1-x_2-x_3))$
G_3	1	u_{1234}	$-x_1x_2x_3x_4$	$\min_y(y(3-x_1-x_2-x_3-x_4))$
G_4	1	$l'_{\emptyset}$	$-x_1x_2x_3x_4+x_1x_2x_3+x_1x_2x_4$ $+x_1x_3x_4+x_2x_3x_4-x_1x_2$ $-x_1x_3-x_1x_4-x_2x_3-x_2x_4-x_3x_4$	$\min_y(y(1-x_1-x_2-x_3-x_4))$
G_5	4	$l'_{\{1,2,3\}}$	$x_1x_2x_3x_4-x_1x_2x_3$ $-x_1x_4-x_2x_4-x_3x_4$	$\min_y(y(2-x_1-x_2-x_3-2x_4))$
G_6	4	$u_{\{12,13,23\}}$	$x_1x_2x_3-x_1x_2-x_1x_3-x_2x_3$	$\min_y(y(1-x_1-x_2-x_3))$
G_7	4	$u_{\{123,124,134\}}$	$x_1x_2x_3x_4-x_1x_2x_3-x_1x_2x_4$ $-x_1x_3x_4$	$\min_y(y(3-2x_1-x_2-x_3-x_4))$
G_8	1	$u_{\{123,124,134,234\}}$	$2x_1x_2x_3x_4-x_1x_2x_3-x_1x_2x_4$ $-x_1x_3x_4-x_2x_3x_4$	$\min_y(y(2-x_1-x_2-x_3-x_3))$
G_9 :	6	$u_{\{12,134,234\}}$	$x_1x_2x_3x_4-x_1x_2-x_1x_3x_4-x_2x_3x_4$	$\min_{y_1,y_2}(y_1+2y_2-y_1y_2$ $-y_1x_1-y_1x_2-y_2x_3-y_2x_4)$
G_{10}	6	qin'_{12}	$-x_1x_2x_3x_4+x_1x_3x_4+x_2x_3x_4$ $-x_1x_3-x_1x_4-x_2x_3-x_2x_4-x_3x_4$	

Figure 2: Examples of generators of $\Gamma_{\mathsf{sub},4}$, with corresponding polynomials and gadgets.

The gadgets shown in Figure 2 can easily be verified, and hence we have established the following:

Proposition 4.3. $G_0, G_1, \ldots G_9 \subseteq \langle \Gamma_{\mathsf{sub},2} \rangle$.

Now we prove that these gadget are optimal in terms of the number of hidden variables. First we show that some cost functions need an extra variable to be expressed over $\Gamma_{\mathsf{sub},2}$.

Proposition 4.4. *Any gadget for a cost function $f \in \bigcup_{2\le i\le 8} G_i$ needs at least one extra variable.*

Proof. The proof is a simple case analysis. We show the idea of the proof for G_2. Let $f \in G_2$, for instance, $f = u_{123}$. The polynomial corresponding to $f(x_1, x_2, x_3, x_4)$ is $-x_1x_2x_3$. Assume f can be expressed over $\Gamma_{\mathsf{sub},2}$ as $p(x_1, x_2, x_3, x_4) = a_0 + \sum_{i=1}^{4} a_ix_i + \sum_{1\le i<j\le 4} a_{ij}x_ix_j$, where $a_{ij} \le 0$ (by Corollary 2.5). From the definition of f, clearly $a_0 = a_i = 0$ for $1 \le i \le 4$. Moreover, from the definition of f, $a_{ij} = 0$ for every $1 \le i < j \le 4$. But then $p(1, 1, 1, 0) = 0$ but $f(1, 1, 1, 0) = 1$, which shows that p does not express f. A similar argument works for each other class G_i, with $3 \le i \le 8$. $\square$

Next we show that cost functions from G_9 need two extra variables to be expressed over $\Gamma_{\mathsf{sub},2}$.

Proposition 4.5. *Any gadget for a cost function $f \in G_9$ needs at least two extra variables.*

Proof. The proof is a case analysis. We show the idea of the proof for $f \in G_9$, where $f = u_{\{12,134,234\}}$. The polynomial corresponding to $f(x_1, x_2, x_3, x_4)$ is $x_1x_2x_3x_4 - x_1x_2 - x_1x_3x_4 - x_2x_3x_4$.

Assume f can be expressed over $\Gamma_{\mathsf{sub},2}$ using only one extra variable as

$$p(x_1, x_2, x_3, x_4) = a_0 + \sum_{i=1}^{4} a_i x_i + \sum_{1 \le i < j \le 4} a_{ij} x_i x_j + \min_y (y(c + \sum_{i=1}^{4} b_i x_i)),$$

where $a_{ij}, b_i \le 0$ by Corollary 2.5. From the definition of f, $a_0 = 0$, and also $a_{ij} = 0$ except for a_{12}.

First assume that $a_{12} = 0$, that is, $p = \sum_{i=1}^{4} a_i x_i + \min_y (y(c + \sum_{i=1}^{4} b_i x_i))$. In other words, $p = \sum_{i=1}^{4} a_i x_i + \min(0, c + \sum_{i=1}^{4} b_i x_i)$. From the definition of f, $a_i \ge 0$ for $1 \le i \le 4$, and

$$\begin{array}{rcl} a_1 + a_2 + a_3 + c + b_1 + b_2 + b_3 & = & -1 \\ a_1 + a_2 + a_4 + c + b_1 + b_2 + b_4 & = & -1 \\ a_1 + a_3 + a_4 + c + b_1 + b_3 + b_4 & = & -1 \\ a_2 + a_3 + a_4 + c + b_2 + b_3 + b_4 & = & -1 \\ a_1 + a_2 + a_3 + a_4 + c + b_1 + b_2 + b_3 + b_4 & = & -2 \end{array}$$

It follows, that $a_i + b_i = -1$ for $1 \le i \le 4$. Hence, $p = \sum_{i=1}^{4} a_i x_i + \min(0, c + \sum_{i=1}^{4}(-a_i - 1)x_i)$. As $f(1,1,0,0) = -1$, $c = 1$. Therefore, $p(1,1,1,0) = -2 \ne -1 = f(1,1,1,0)$, and clearly p does not express f.

Now assume that $a_{12} < 0$. As $f(1,1,0,0) = -1$, $a_{12} = -1$. From the definition of f,

$$\begin{array}{rcl} a_{12} + a_1 + a_2 + a_3 + \min(0, c + b_1 + b_2 + b_3) & = & -1 \\ a_{12} + a_1 + a_2 + a_4 + \min(0, c + b_1 + b_2 + b_4) & = & -1 \\ a_1 + a_3 + a_4 + \min(0, c + b_1 + b_3 + b_4) & = & -1 \\ a_2 + a_3 + a_4 + \min(0, c + b_2 + b_3 + b_4) & = & -1 \\ a_{12} + a_1 + a_2 + a_3 + a_4 + \min(0, c + b_1 + b_2 + b_3 + b_4) & = & -2 \end{array}$$

which can be simplified as

$$\begin{array}{rcl} a_1 + a_2 + a_3 + \min(0, c + b_1 + b_2 + b_3) & = & 0 \\ a_1 + a_2 + a_4 + \min(0, c + b_1 + b_2 + b_4) & = & 0 \\ a_1 + a_3 + a_4 + c + b_1 + b_3 + b_4 & = & -1 \\ a_2 + a_3 + a_4 + c + b_2 + b_3 + b_4 & = & -1 \\ a_1 + a_2 + a_3 + a_4 + c + b_1 + b_2 + b_3 + b_4 & = & -1 \end{array}$$

It follows, that $a_1 + b_1 = 0$ and $a_2 + b_2 = 0$. Now if $a_1 = a_2 = 0$, then $b_1 = b_2 = 0$ and $p = -1 + a_3 + a_4 + \min(0, c + b_3x_3 + b_4x_4)$. As $p(0,0,1,1) = p(1,1,1,1)$, but

$f(0,0,1,1) = -1 \neq -2 = f(1,1,1,1)$, p does not express f. Since we have $a_1 > 0$ or $a_2 > 0$, we get

$$\begin{array}{rcr} a_1 + a_2 + a_3 + c + b_1 + b_2 + b_3 &=& 0 \\ a_1 + a_2 + a_4 + c + b_1 + b_2 + b_4 &=& 0 \\ a_1 + a_3 + a_4 + c + b_1 + b_3 + b_4 &=& -1 \\ a_2 + a_3 + a_4 + c + b_2 + b_3 + b_4 &=& -1 \\ a_1 + a_2 + a_3 + a_4 + c + b_1 + b_2 + b_3 + b_4 &=& -1 \end{array}$$

It follows, that $a_3 + b_3 = -1$ and $a_4 + b_4 = -1$. Hence, $p = -1 + \sum_{i=1}^{4} a_i x_i + \min(0, c - a_1 x_1 - a_2 x_2 + (-a_3 - 1)x_3 + (-a_4 - 1)x_4)$. As $f(1,1,0,0) = -1$, $c = 2$. However, $p(0,1,1,1) = 0 \neq -1 = f(0,1,1,1)$, and therefore p does not express f. $\square$

It is not difficult to check that for any cost functions $f, g \in \bigcup_{2 \leq i \leq 8} G_i$, the gadgets for f and g can use the same extra variable. Similarly, for any $f, g \in G_9$, the gadgets for f and g can use the same extra two variables. The key observation is that for any two cost functions $f, g \in G_i$, the gadgets for f and g from Figure 2 have the same minimal assignments to the extra variables[6].

Therefore, we need only one extra variable for any $f \in \mathcal{L}(G_i)$, $2 \leq i \leq 8$ and two for any $f \in \mathcal{L}(G_9)$. This gives us an upper bound of $7 + 2 = 9$ extra variables per 4-ary constraint from $\mathcal{L}(\bigcup_{i=0}^{9} G_i)$. Note that this is sufficient[7] for the claim of Corollary 4.2.

Next we show using Theorem 2.4 that no $f \in G_{10}$ is expressible over $\Gamma_{\mathsf{sub},2}$. In order to do so we first characterise the multimorphisms of Γ_{cut}.

A function $\mathcal{F} : D^k \to D^k$ is called *conservative* if, for each possible choice of $x_1, \ldots, x_k$, the tuple $\mathcal{F}(x_1, \ldots, x_k)$ contains the same multi-set of values $x_1, \ldots, x_k$ (in some order).

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, we denote by $H(\mathbf{x}, \mathbf{y})$ the *Hamming distance* between $\mathbf{x}$ and $\mathbf{y}$, which is the number of positions for which the corresponding values are different.

Theorem 4.6. $\mathcal{F} \in \mathsf{Mul}(\Gamma_{\mathsf{cut}}) \Leftrightarrow \mathcal{F}$ *is conservative and Hamming distance non-increasing.*

Proof. Recall that Γ_{cut} consists of unary cost functions μ_u^c, for $c \in \overline{\mathbb{Q}}_+$ and $u \in \{0,1\}$, and binary cost functions χ^w, for $w \in \overline{\mathbb{Q}}_+$.

First we show that for every $w \in \overline{\mathbb{Q}}_+$, χ^w can be equivalently replaced by the following binary cost function

$$=_w (x,y) \;=\; \begin{cases} 0 & \text{if } x = y, \\ w & \text{otherwise,} \end{cases}$$

without changing the expressive power of Γ_{cut}.

[6]More formally, for any assignment of x_1, x_2, x_3 and x_4, the minimal assignment to the extra variables in the gadget for f is the same as the minimal assignment to the extra variables in the gadget for g.

[7]However, a better upper bound of only 3 extra variables per constraint can be obtained by a more careful analysis of the minimal assignments. Note that this is almost optimal as we have proved a lower bound of 2 extra variables.

Let $w \in \overline{\mathbb{Q}}_+$ be fixed,

$$\mathcal{P}_1 = \langle \{x, y\}, \{0, 1\}, \{\langle \langle x, y\rangle, \chi^w\rangle, \langle \langle y, x\rangle, \chi^w\rangle\}\rangle,$$

and

$$\mathcal{P}_2 = \langle \{x, y\}, \{0, 1\}, \{\langle \langle x, y\rangle, =_w\rangle, \langle x, \mu_1^w\rangle, \langle y, \mu_0^w\rangle\}\rangle.$$

Then, $\langle \mathcal{P}_1, \{x, y\}\rangle$ is a gadget for expressing $=_w$ over $\{\chi^w\}$, and $\langle \mathcal{P}_2, \{x, y\}\rangle$ is a gadget for expressing $\chi^{2w} + w$ over $\{=_w, \mu_0^w, \mu_1^w\}$.

Let $\mathcal{F} \in \mathsf{Mul}(\Gamma_{\mathsf{cut}})$ where $\mathcal{F} : D^k \to D^k$. Assume that $\mathcal{F}$ is not conservative, that is, 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 $\langle u_1, \dots, u_k\rangle$. Then $\mathcal{F}$ cannot be a multimorphism of the unary cost function $\mu_{v_i}^1$. On the other hand, any conservative $\mathcal{F}$ is clearly a multimorphism of μ_u^c for every $c \in \overline{\mathbb{Q}}_+$ and $u \in D$.

By Definition 2.3, $\mathcal{F}$ is a multimorphism of $=_w$ if and only if the following holds:

$$H(\langle u_1, \dots, u_k\rangle, \langle v_1, \dots, v_k\rangle) \geq H(\mathcal{F}(u_1, \dots, u_k), \mathcal{F}(v_1, \dots, v_k)).$$

$\square$

Now consider the function $\mathcal{F}_{sep} : \{0, 1\}^5 \to \{0, 1\}^5$ defined in Figure 3. (A definition of $\mathcal{F}_{sep}$ in a different format can be found in Figure 5 in Appendix A).

	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
x	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
	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	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
$\mathcal{F}_{sep}(x)$	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
	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
	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

Figure 3: Definition of $\mathcal{F}_{sep}$.

Although it is rather tedious, the following can be checked:

Proposition 4.7. *$\mathcal{F}_{sep}$ is conservative and Hamming distance non-increasing.*

Proposition 4.8. *$\mathcal{F}_{sep} \notin \mathsf{Mul}(\{qin'_{12}\})$.*

Proof. See Figure 4. $\square$

It follows that $\mathcal{F}_{sep}$ is *not* a multimorphism of any $f \in G_{10}$, so, by Theorem 2.4, we have.

Corollary 4.9. *No $f \in G_{10}$ is expressible over $\Gamma_{\mathsf{sub},2}$.*

$$\mathcal{F}_{sep}\quad \begin{array}{cccc} 1&0&1&0\\ 1&0&0&1\\ 0&1&0&1\\ 0&1&1&0\\ 0&0&1&1\\ \hline 0&0&1&0\\ 0&0&0&1\\ 1&1&0&0\\ 1&0&1&1\\ 0&1&1&1 \end{array} \quad \begin{array}{c} \xrightarrow{qin'_{12}} \\[10em] \xrightarrow{qin'_{12}} \end{array} \quad \begin{array}{l} \left.\begin{array}{r} -1\\ -1\\ -1\\ -1\\ -1 \end{array}\right\} \sum = \text{-5} \\ \left.\begin{array}{r} 0\\ 0\\ 0\\ -2\\ -2 \end{array}\right\} \sum = \text{-4} \end{array}$$

Figure 4: $\mathcal{F}_{sep} \notin \mathsf{Mul}(\{qin'_{12}\})$.

Proof. First note that from the (in)expressibility point of view there is no difference between qin_a and qin'_a as constant and unary cost functions are trivial. Let $f \in G_{10}$, for instance, $f = qin_{12}$. By Proposition 4.8, $\mathcal{F}_{sep}$ is not a multimorphism of qin'_{12}. However, by Proposition 4.7 and Theorem 4.6, $\mathcal{F}_{sep} \in \mathsf{Mul}(\Gamma_{\mathsf{cut}})$. Hence, by Theorem 2.4, qin'_{12} is not expressible over Γ_{cut}, so qin'_{12} is not expressible over $\Gamma_{\mathsf{sub},2}$. Consequently, qin_{12} is also not expressible over $\Gamma_{\mathsf{sub},2}$.

Note that the same proof works for any $f \in G_{10}$: any $f \in G_{10}$ is of the form $f = qin_a$, where a is a 2-element subset of $\{1, 2, 3, 4\}$. The table in Figure 4 with correspondingly permuted columns[8] shows that $\mathcal{F}_{sep} \notin \mathsf{Mul}(\{qin'_a\})$ for any $qin_a \in G_{10}$. $\square$

This completes the proof of Theorem 4.1.

Finally, we show that the question of whether a 4-ary submodular cost function is expressible using binary submodular cost functions is checkable efficiently.

Definition 4.10. *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 (C) if for each $\{i, j\}, \{k, l\} \subset \{1, 2, 3, 4\}$, with i, j, k, l distinct, we have $a_{\{i,j\}} + a_{\{k,l\}} + a_{\{i,j,k\}} + a_{\{i,j,l\}} \leq 0$.*

Proposition 4.11.

- *Every $f \in \bigcup_{i=0}^{9} G_i$ satisfies condition (C).*
- *No $f \in G_{10}$ satisfies condition (C).*
- *If both f and g satisfy condition (C), and $\alpha \geq 0$, then both αf and $f + g$ satisfy (C) as well.*

Proof. The first statement is easily verified by a simple case analysis (see polynomials in Figure 2).The second statement is true because $a_{\{1,2\}} + a_{\{3,4\}} + a_{\{1,2,3\}} + a_{\{1,2,4\}} =$

[8]The first two columns are on the i-th and j-th position in the case of the function $f = qin'_a$ where $a = \{i, j\}$.

$0-1+1+1=1 \not\leq 0$ and therefore condition (C) is violated for $i=1$ and $j=2$. The third statement is easy: (C) is preserved under addition of polynomials and multiplying a polynomial by a non-negative number. $\square$

Corollary 4.12. *For any $f \in \Gamma_{\mathsf{sub},4}$, condition (C), described in Definition 4.10, can be used to efficiently test whether or not $f \in \langle \Gamma_{\mathsf{sub},2} \rangle$.*

5 Conclusion

In this paper we studied the question of which submodular constraints are *expressible* using binary submodular constraints, and hence can be minimised efficiently using an algorithm to find a minimum cut in a suitable graph.

First we showed that in order to answer this question it is sufficient to consider the Boolean finite-valued case only. We then focused on 4-ary submodular functions and showed that 9 out of 10 classes of generators for 4-ary submodular functions are expressible using binary submodular functions. Therefore, any 4-ary submodular constraint which can be expressed as a non-negative combination of generators from those 9 classes can be minimised efficiently using minimum cuts. Using algebraic properties of valued constraints, we showed that the remaining class of generators is *not* expressible.

Finally, we showed how to test efficiently whether a given 4-ary submodular constraint is expressible using binary submodular constraints by examining its polynomial representation.

Acknowledgements The authors would like to thank David Cohen and Martin Cooper for fruitful discussions on submodular constraints.

References

[1] Billionet, A., Minoux, M.: Maximizing a supermodular pseudo-boolean function: a polynomial algorithm for cubic functions. Discrete Applied Mathematics **12** (1985) 1–11 2, 8

[2] Bistarelli, S., Fargier, H., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G.: Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. Constraints **4** (1999) 199–240 1

[3] Boros, E., Hammer, P.L.: Pseudo-boolean optimization. Discrete Applied Mathematics **123**(1-3) (2002) 155–225 2, 7, 8

[4] Bulatov, A., Krokhin, A., Jeavons, P.: Classifying the complexity of constraints using finite algebras. SIAM Journal on Computing **34**(3) (2005) 720–742 5

[5] Cohen, D., Cooper, M., Jeavons, P.: An algebraic characterisation of complexity for valued constraints. In: CP'06. Volume 4204 of LNCS. (2006) 107–121 5

[6] Cohen, D., Cooper, M., Jeavons, P.: Generalising submodularity and Horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. Theoretical Computer Science **401** (2008) 36–51 2

[7] Cohen, D., Cooper, M., Jeavons, P., Krokhin, A.: A maximal tractable class of soft constraints. Journal of Artificial Intelligence Research **22** (2004) 1–22 2

[8] Cohen, D., Cooper, M., Jeavons, P., Krokhin, A.: Supermodular functions and the complexity of Max-CSP. Discrete Applied Mathematics **149** (2005) 53–72 9

[9] Cohen, D., Cooper, M., Jeavons, P., Krokhin, A.: The complexity of soft constraint satisfaction. Artificial Intelligence **170** (2006) 983–1016 1, 2, 6, 8

[10] Cooper, M.C.: Minimization of locally defined submodular functions by optimal soft arc consistency. Constraints **13** (2008) 2

[11] Cooper, M.: High-order consistency in valued constraint satisfaction. Constraints **10** (2005) 283–305 4

[12] Creignou, N., Khanna, S., Sudan, M.: Complexity Classification of Boolean Constraint Satisfaction Problems. Volume 7 of SIAM Monographs on Discrete Mathematics and Applications. SIAM (2001) 1, 9

[13] Deineko, V., Jonsson, P., Klasson, M., Krokhin, A.: The approximability of Max CSP with fixed-value constraints. Journal of the ACM (2008) (accepted). 2

[14] Freuder, E.C.: Complexity of k-tree structured constraint satisfaction problems. In: AAAI. (1990) 4–9 1

[15] Fujishige, S.: Submodular Functions and Optimization. 2nd edn. Volume 58 of Annals of Discrete Mathematics. North-Holland, Amsterdam (2005) 2

[16] Gottlob, G., Leone, L., Scarcello, F.: A comparison of structural CSP decomposition methods. Artificial Intelligence **124** (2000) 243–282 1

[17] Gottlob, G., Leone, L., Scarcello, F.: Hypertree decomposition and tractable queries. Journal of Computer and System Sciences **64**(3) (2002) 579–627 1

[18] Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. Journal of the ACM **54**(1) (2007) 1

[19] Gutin, G., Rafiey, A., Yeo, A.: Minimum cost and list homomorphisms to semicomplete digraphs. Discrete Applied Mathematics **154** (2006) 890–897 1

[20] Iwata, S.: Submodular function minimization. Mathematical Programming **112** (2008) 45–64 2

[21] Jeavons, P., Cohen, D., Cooper, M.: Constraints, consistency and closure. Artificial Intelligence **101**(1–2) (1998) 251–265 7

[22] Jonsson, P., Klasson, M., Krokhin, A.: The approximability of three-valued MAX CSP. SIAM Journal on Computing **35**(6) (2006) 1329–1349 2

[23] Kolmogorov, V., Zabih, R.: What energy functions can be minimized via graph cuts? IEEE Transactions on Pattern Analysis and Machine Intelligence **26**(2) (2004) 147–159 3

[24] Krokhin, A., Larose, B.: Maximizing supermodular functions on product lattices, with application to maximum constraint satisfaction. SIAM Journal on Discrete Mathematics **22**(1) (2008) 312–328 2

[25] Montanari, U.: Networks of constraints: Fundamental properties and applications to picture processing. Information Sciences **7** (1974) 95–132 1

[26] Nemhauser, G., Wolsey, L.: Integer and Combinatorial Optimization. John Wiley & Sons (1988) 2, 6

[27] Orlin, J.B.: A faster strongly polynomial time algorithm for submodular function minimization. In: IPCO'07. Volume 4513 of LNCS. (2007) 240–251 2

[28] Promislow, S., Young, V.: Supermodular functions on finite lattices. Order **22**(4) (2005) 389–413 3, 9, 10, 11

[29] Rhys, J.: A selection problem of shared fixed costs and network flows. Management Science **17**(3) (1970) 200–207 9

[30] Rossi, F., van Beek, P., Walsh, T., eds.: The Handbook of Constraint Programming. Elsevier (2006) 1

[31] Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: hard and easy problems. In: IJCAI'95. (1995) 1, 4

[32] Schrijver, A.: A combinatorial algorithm minimizing submodular functions in strongly polynomial time. Journal of Combinatorial Theory, Series B **80** (2000) 346–355 2

[33] Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Volume 24 of Algorithms and Combinatorics. Springer (2003) 2, 6

[34] Topkis, D.: Supermodularity and Complementarity. Princeton University Press (1998) 2

[35] Živný, S., Jeavons, P.G.: Classes of submodular constraints expressible by graph cuts. In: CP'08. (2008) (to appear). 2, 8, 9

[36] Zalesky, B.: Efficient determination of gibbs estimators with submodular energy functions. arXiv:math/0304041v1 (February 2008) 3

A $\mathcal{F}_{sep}$

x	$\mathcal{F}_{sep}(x)$
00000	00000
00001	00001
00010	00001
00011	00011
00100	00001
00101	00011
00110	00101
00111	00111
01000	00010
01001	00011
01010	00011
01011	00111
01100	00011
01101	01011
01110	00111
01111	01111
10000	00010
10001	00011
10010	00011
10011	10011
10100	00011
10101	00111
10110	00111
10111	10111
11000	00110
11001	00111
11010	00111
11011	10111
11100	00111
11101	01111
11110	01111
11111	11111

Figure 5: Definition of $\mathcal{F}_{sep}$ written differently.

B Generators of 4-ary Boolean submodular cost functions

To simplify notation in the following figures, we omit curly brackets, which are normally used to denote sets. For instance, we write 123 instead of $\{1, 2, 3\}$. We denote by [4] the set $\{1, 2, 3, 4\}$.

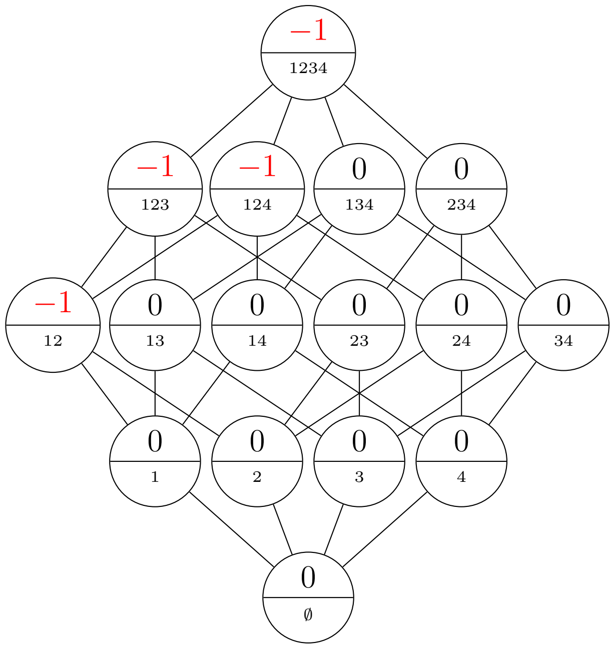

Figure 6: G_1: u_x, where $x \subseteq [4]$, $|x| = 2$; here $x = \{1, 2\}$.

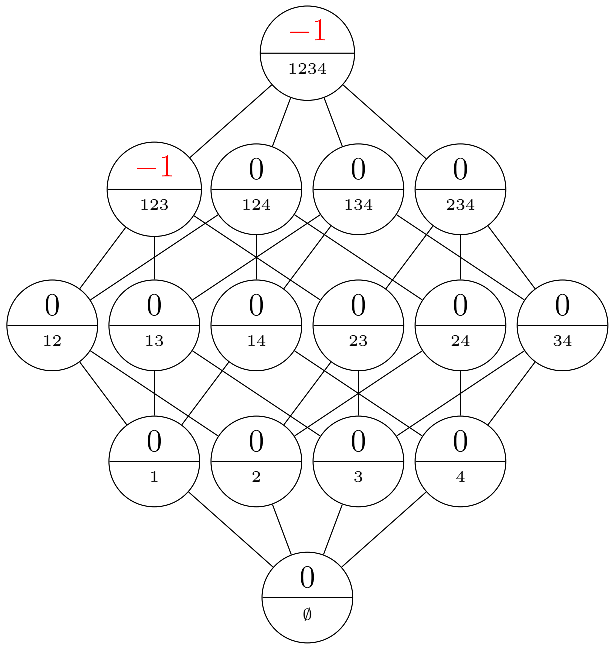

Figure 7: G_2: u_x, where $x \subseteq [4]$, $|x| = 3$; here $x = \{1, 2, 3\}$.

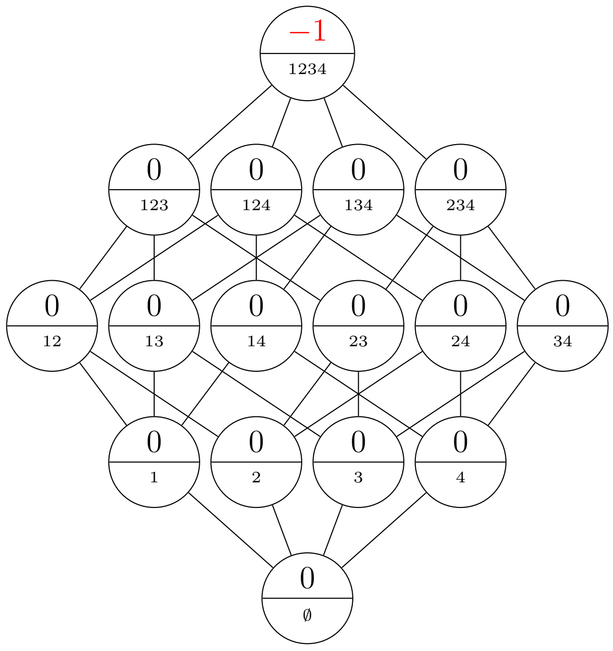

Figure 8: G_3: u_x, where $x \subseteq [4]$, $|x| = 4$; $x = [4]$.

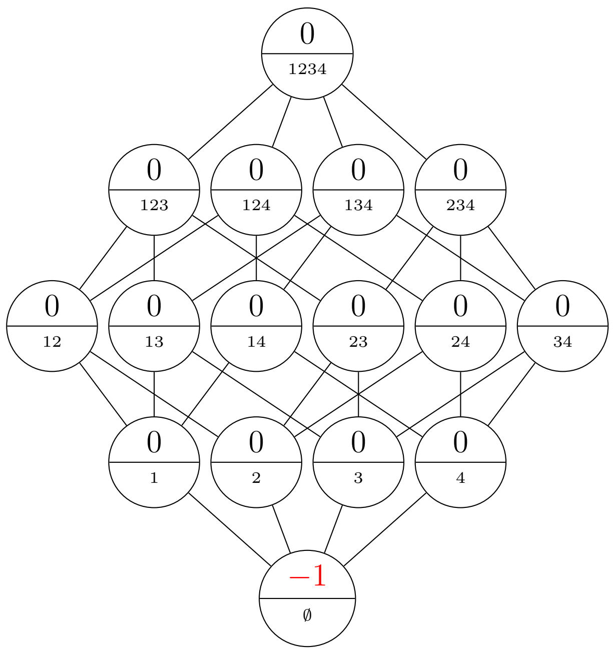

Figure 9: G_4: $l_\emptyset$.

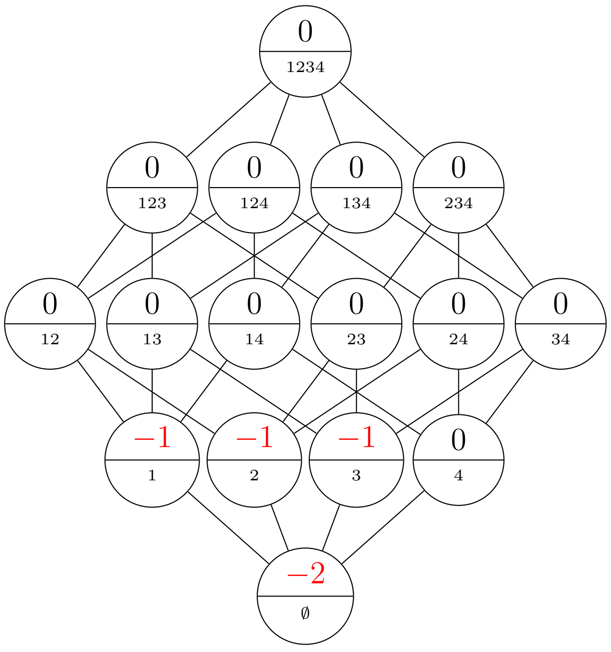

Figure 10: G_5: l_B, where B consists of three 1-element subsets of $[4]$; here $B = \{1, 2, 3\}$.

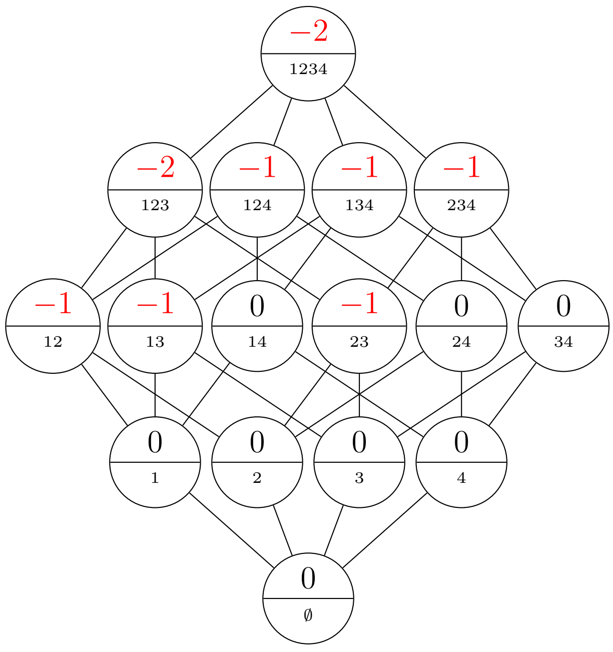

Figure 11: G_6: u_F, where F consists of three 2-element subsets of any 3-element subset of $[4]$; here $F = \{12, 13, 23\}$.

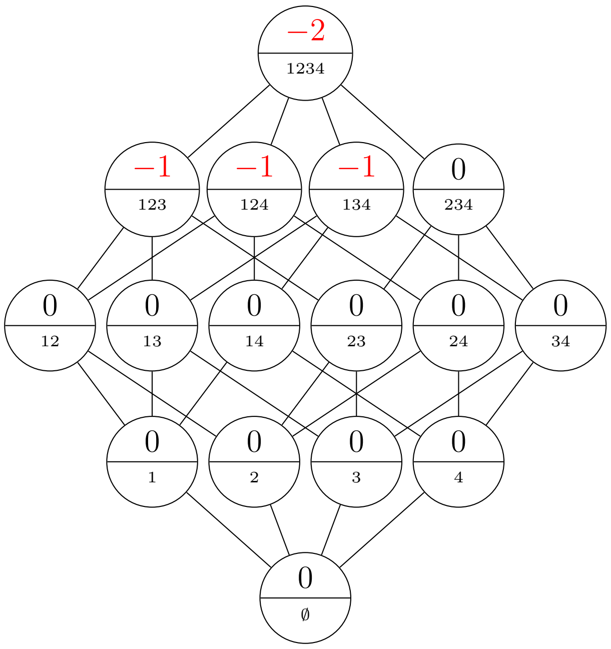

Figure 12: G_7: u_F, where F consists of three 3-element subsets of $[4]$; here $F = \{123, 124, 134\}$.

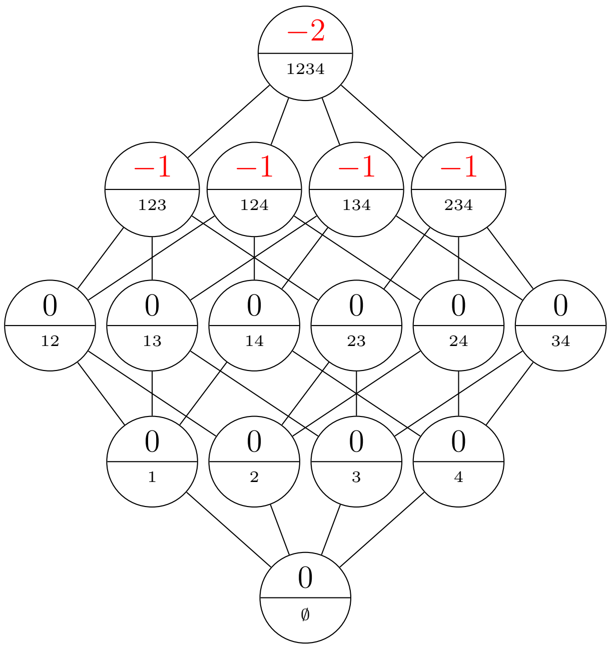

Figure 13: G_8: u_F, where F consists of all four 3-element subsets of $[4]$; $F = \{123, 124, 134, 234\}$.

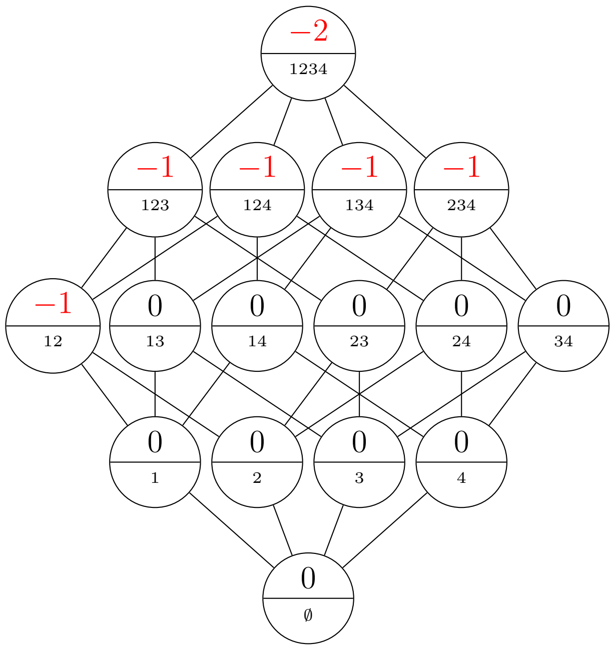

Figure 14: G_9: u_F, where F consists of two 3-element subsets of [4] and the corresponding (symmetric difference) subset of [4] of size two; here $F = \{134, 234, 12\}$.

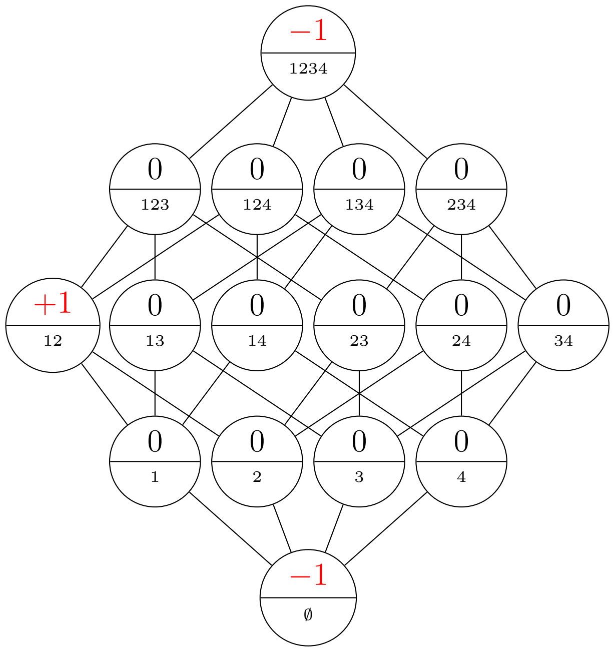

Figure 15: G_{10}: qin_a, where a is a 2-element subset of [4]; here $a = \{12\}$.

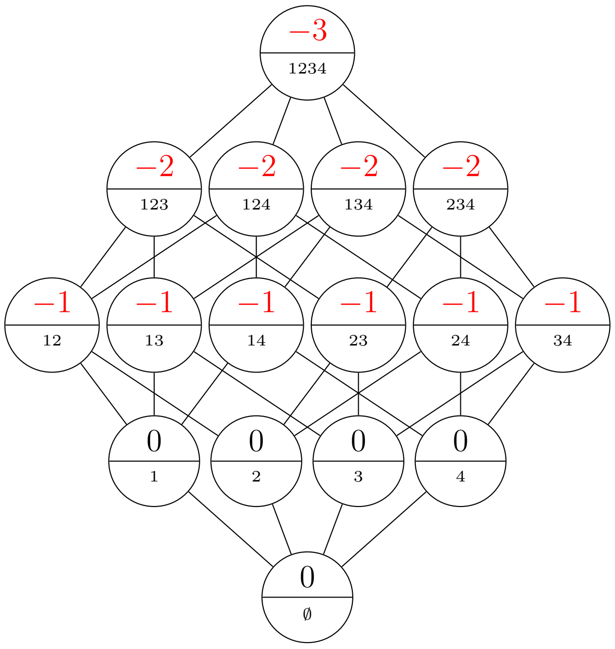

Figure 16: G_4: $l'_\emptyset$, equivalent to $l_\emptyset$ from Figure 9.

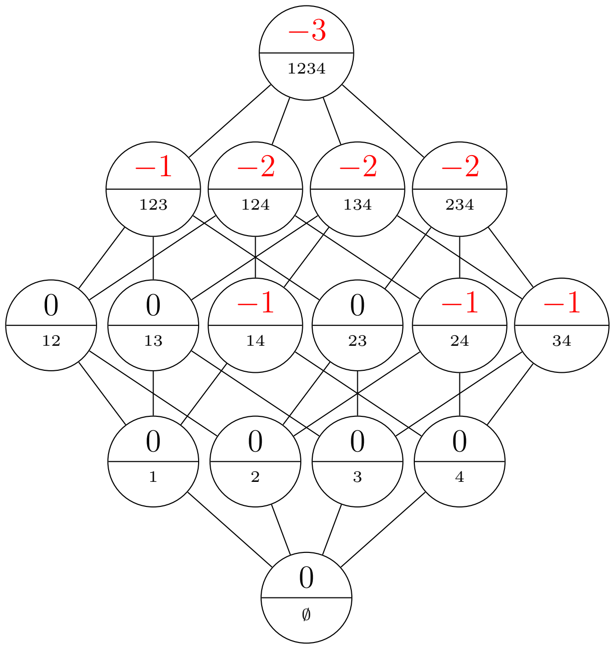

Figure 17: G_5: l'_B, equivalent to l_B from Figure 10, $B = \{1, 2, 3\}$.

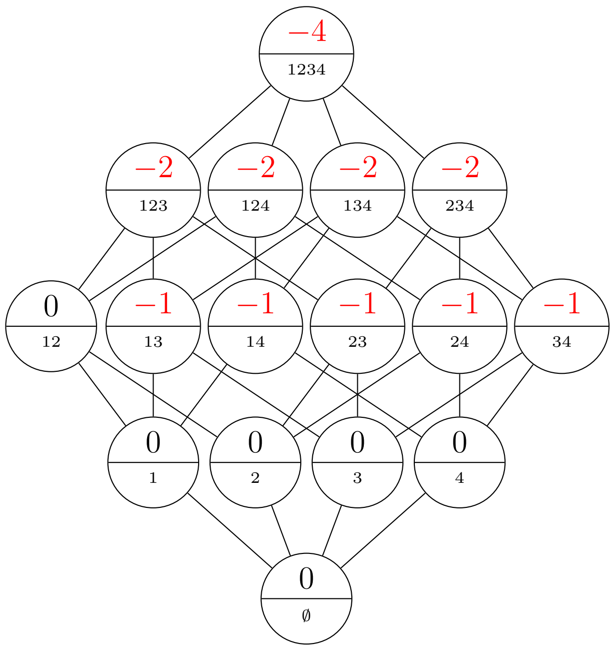

Figure 18: G_{10}: qin'_a, equivalent to qin_a from Figure 15, $a = \{12\}$.