The approximability of Max CSP with fixed-value constraints

Vladimir Deineko
Warwick Business School
University of Warwick, UK
Vladimir.Deineko@wbs.ac.uk

Peter Jonsson
Dep't of Computer and Information Science
University of Linköping, Sweden
peter.jonsson@ida.liu.se

Mikael Klasson
Dep't of Computer and Information Science
University of Linköping, Sweden
mikael.klasson@ida.liu.se

Andrei Krokhin
Department of Computer Science
University of Durham, UK
andrei.krokhin@durham.ac.uk

Abstract

In the maximum constraint satisfaction problem (Max CSP), one is given a finite collection of (possibly weighted) constraints on overlapping sets of variables, and the goal is to assign values from a given finite domain to the variables so as to maximize the number (or the total weight, for the weighted case) of satisfied constraints. This problem is **NP**-hard in general, and, therefore, it is natural to study how restricting the allowed types of constraints affects the approximability of the problem. In this paper, we show that any Max CSP problem with a finite set of allowed constraint types, which includes all fixed-value constraints (i.e., constraints of the form $x = a$), is either solvable exactly in polynomial time or else is **APX**-complete, even if the number of occurrences of variables in instances is bounded. Moreover, we present a simple description of all polynomial-time solvable cases of our problem. This description relies on the well-known algebraic combinatorial property of supermodularity.

Keywords: maximum constraint satisfaction, complexity of approximation, dichotomy, supermodularity, Monge properties

1 Introduction and Related Work

1.1 Background

Many combinatorial optimization problems are **NP**-hard, and the use of approximation algorithms is one of the most prolific techniques to deal with **NP**-hardness. However, hard optimization problems exhibit different behaviour with respect to approximability, and complexity theory for approximation is now a well-developed area [2].

Constraint satisfaction problems (CSPs) have always played a central role in this direction of research, since the CSP framework contains many natural computational problems, for example, from propositional logic and graph theory (see, e.g., [13, 25]). In a CSP, informally speaking, one is given a finite collection of constraints on overlapping sets of variables, and the goal is to decide whether there is an assignment of values from a given domain to the variables satisfying all constraints (decision problem) or to find an assignment satisfying maximum number of constraints (optimization problem). These are the main versions of the CSP, and there are many other versions obtained from them by modifying the objective (see, e.g., [13, 34]). In this paper, we will focus on

the optimization problems, which are known as *maximum constraint satisfaction* problems, MAX CSP for short. The most well-known examples of such problems are MAX k-SAT and MAX CUT. Let us now formally define MAX CSP.

Let D denote a *finite* set with $|D| > 1$. Let $R_D^{(m)}$ denote the set of all m-ary predicates over D, that is, functions from D^m to $\{0,1\}$, and let $R_D = \bigcup_{m=1}^{\infty} R_D^{(m)}$. Also, let $\mathbb{Z}^+$ denote the set of all non-negative integers.

Definition 1.1 (constraint) *A* constraint *over a set of variables $V = \{x_1, x_2, \ldots, x_n\}$ is an expression of the form $f(\mathbf{x})$ where*

- *$f \in R_D^{(m)}$ is called the* constraint predicate*; and*
- *$\mathbf{x} = (x_{i_1}, \ldots, x_{i_m})$ is called the* constraint scope*.*

The constraint $f(\mathbf{x})$ is said to be satisfied *on a tuple $\mathbf{a} = (a_{i_1}, \ldots, a_{i_m}) \in D^m$ if $f(\mathbf{a}) = 1$.*

Note that throughout the paper the values 0 and 1 taken by any predicate will be considered as integers, not as Boolean values, and addition will always denote the addition of integers.

Definition 1.2 (MAX CSP**)** *For a finite $\mathcal{F} \subseteq R_D$, an instance of* MAX CSP($\mathcal{F}$) *is a pair (V, C) where*

- *$V = \{x_1, \ldots, x_n\}$ is a set of variables taking their values from the set D;*
- *C is a collection of constraints $f_1(\mathbf{x}_1), \ldots, f_q(\mathbf{x}_q)$ over V, where $f_i \in \mathcal{F}$ for all $1 \leq i \leq q$.*

The goal is to find an assignment $\varphi : V \to D$ that maximizes the number of satisfied constraints, that is, to maximize the function $f : D^n \to \mathbb{Z}^+$, defined by $f(x_1, \ldots, x_n) = \sum_{i=1}^{q} f_i(\mathbf{x}_i)$. If the constraints have (positive integral) weights ϱ_i, $1 \leq i \leq q$, then the goal is to maximize the total weight of satisfied constraints, that is, to maximize the function $f : D^n \to \mathbb{Z}^+$, defined by $f(x_1, \ldots, x_n) = \sum_{i=1}^{q} \varrho_i \cdot f_i(\mathbf{x}_i)$.

Complexity classifications for various versions of constraint satisfaction problems have attracted much attention in the recent years (see survey [34]) because, as the authors of [13] nicely put it, these classifications "present a reasonably accurate bird's eye view of computational complexity and the equivalence classes it has created". Classifications with respect to a set of allowed constraint types (such as $\mathcal{F}$ in MAX CSP($\mathcal{F}$) above) have been of particular interest, e.g.,[5, 6, 8, 9, 13, 17, 24].

Boolean constraint satisfaction problems (that is, when $D = \{0,1\}$) are by far better studied [13] than the non-Boolean version. The main reason is, in our opinion, that Boolean constraints can be conveniently described by propositional formulas which provide a flexible and easily manageable tool, and which have been extensively used in complexity theory from its very birth. Moreover, Boolean CSPs suffice to represent a number of well-known problems and to obtain results clarifying the structure of complexity for large classes of interesting problems [13]. In particular, Boolean CSPs were used to provide evidence for one of the most interesting phenomena in complexity theory, namely that interesting problems belong to a small number of complexity classes [13], which cannot be taken for granted due to Ladner's theorem. After the pioneering work of Schaefer [38] presenting a tractable versus **NP**-complete dichotomy for Boolean decision CSPs, many classification results have been obtained (see, e.g., [13]), most of which are dichotomies. In particular, a dichotomy in complexity and approximability for Boolean MAX CSP has been obtained by Creignou [12], and

it was slightly refined in [31] (see also [13]). The complexity of Boolean MAX CSP with arbitrary (i.e., not necessarily positive) weights was classified in [27].

Many papers on various versions of Boolean CSPs mention studying non-Boolean CSPs as a possible direction of future research, and additional motivation for it, with an extensive discussion, was given by Feder and Vardi [17]. Dichotomy results on non-Boolean CSPs give a better understanding of what makes a computational problem tractable or hard, and they give a more clear picture of the structure of complexity of problems, since many facts observed in Boolean CSPs appear to be special cases of more general phenomena. Notably, many appropriate tools for studying non-Boolean CSPs have not been discovered until recently. For example, universal algebra tools have proved to be very fruitful when working with decision, counting, and quantified CSPs [5, 6, 7, 8, 9] while ideas from lattice theory, combinatorial optimization and operations research have been recently suggested for optimization problems [11, 35].

The problem MAX CSP is **NP**-hard in general (i.e., without restrictions on the type of allowed constraints), and there is a significant body of results on algorithmic and complexity-theoretical aspects of this problem, including results on superpolynomial general algorithms (e.g.,[14, 41]), polynomial algorithms for special cases [11, 35], explicit approximability bounds (e.g., [16, 20, 22, 23, 32]), and complexity of approximation (e.g., [3, 13, 28]).

The main research problem that we will look at in this paper is the following one.

Problem 1 *Classify the problems* MAX CSP($\mathcal{F}$) *with respect to approximability.*

We say that a predicate is *non-trivial* if it is not identically 0. We will always assume that $\mathcal{F}$ is finite and contains only non-trivial predicates. Whenever we do not specify which version (weighted or unweighted) we consider, we mean *unweighted* MAX CSP. Note that the definition allows one to repeat constraints in instances (we follow [13] in this), so our unweighted problem actually allows polynomially bounded weights. However, our tractability results will hold for the weighted version, while in our hardness results, for every $\mathcal{F}$, we will use only instances where every constraint occurs at most $k_{\mathcal{F}}$ times (where $k_{\mathcal{F}}$ is a constant depending on $\mathcal{F}$).

For the Boolean case, Problem 1 was solved in [12, 13, 31]. It appears that Boolean MAX CSP($\mathcal{F}$) problems exhibit a dichotomy in that such a problem is either solvable exactly in polynomial time or else **APX**-complete, i.e., does not admit a PTAS (polynomial-time approximation scheme) unless **P**=**NP**. These papers also describe the boundary between the two cases. This dichotomy result was extended to the case $|D| = 3$ in [28]. The complexity of non-Boolean MAX CSP with arbitrary (i.e., not necessarily positive) weights was recently classified in [29].

1.2 Results

For a subset $D' \subseteq D$, let $u_{D'}$ denote the predicate such that $u_{D'}(x) = 1$ if and only if $x \in D'$. Let $\mathcal{U}_D = \{u_{D'} \mid \varnothing \neq D' \subseteq D\}$, that is, $\mathcal{U}_D$ is the set of all non-trivial unary predicates on D. Furthermore, let $\mathcal{C}_D = \{u_{\{d\}} \mid d \in D\}$. Note that predicates from $\mathcal{C}_D$ give rise to constraints of the form $x = d$, i.e., *fixed-value constraints*.

The decision problems CSP($\mathcal{F}$) are similar to MAX CSP($\mathcal{F}$), but the the task is to decide whether all constraints in a given instance can be simultaneously satisfied. Problems of the form CSP($\mathcal{F} \cup \mathcal{U}_D$) are known as conservative (or list) CSPs, and their complexity has been completely classified by Bulatov in [6], while a complexity classification for the problems of the form CSP($\mathcal{F} \cup \mathcal{C}_D$) would imply a classification for all problems CSP($\mathcal{F}$) [9].

In this paper we solve the above Problem 1 for all sets of the form $\mathcal{F} \cup \mathcal{C}_D$ where D is any finite set. (Note that this does not necessarily imply a full solution to Problem 1, as it would

for decision problems.) Our result is parallel to Bulatov's classification of conservative CSPs [6], but our techniques are quite different from the universal-algebraic techniques used in [6]. The universal-algebraic techniques from [6, 9] cannot be applied in the optimization setting because the basic properties of decision CSPs that make these techniques useful are not satisfied by Max CSP.

It was suggested in Section 6 of [11] that Max CSP($\mathcal{F} \cup \mathcal{C}_D$) is solvable exactly in polynomial time if and only if all predicates in $\mathcal{F}$ are supermodular with respect to some linear ordering on D (see definitions in Section 4). We prove that this is indeed the case, and that in all other cases the problem Max CSP($\mathcal{F} \cup \mathcal{C}_D$) is **APX**-complete. Moreover, we show that every **APX**-complete problem of the above form is **APX**-complete even when we further restrict it to instances where the number of occurrences of variables is bounded by some (possibly large) constant. Note that approximability properties for constraint problems with the bounded occurrence property (as well as for related problems on graphs with bounded degree) have been intensively studied in the literature (see, e.g., [1, 4, 21, 30]).

Our classification result uses the combinatorial property of supermodularity which is a well-known source of tractable optimization problems [10, 18, 40], and the technique of strict implementations [13, 31] which allows one to show that an infinite family of problems can express, in a regular way, one of a few basic hard problems. We remark that the idea to use supermodularity in the analysis of the complexity of Max CSP($\mathcal{F}$) is very new, and has not been even suggested in the literature prior to [11]. It was shown in [11, 28] that supermodularity is the *only* source of tractability for problems of the form Max CSP($\mathcal{F}$) when D is small (i.e., $|D| \leq 3$). This, together with the results obtained in the present paper, suggests that supermodularity is indeed the appropriate tool for tackling Problem 1.

Some of our technical results (those in Section 5) are of independent interest in combinatorics. In [33], Klinz *et al.* study how one can permute rows and columns of a 0-1 matrix so as to avoid a collection of given forbidden submatrices; some results of this nature have later been used in constructing phylogenetic trees [36]. Klinz *et al.* obtain many results in this direction, but they leave open the case when matrices are square and rows and columns must be permuted by the *same* permutation (see Section 6 of [33]). Our results clarify the situation in this special case for one type of forbidden matrices considered in Theorem 4.5 of [33].

The structure of the paper is as follows: Section 2 contains definitions of approximation complexity classes and reductions. In Section 3, we describe our reduction techniques, and in Section 4 we give the basics of supermodularity and discuss the relevance of supermodularity in the study of Max CSP. Section 5 contains technical results that are used in the proof of the main classification result of the paper, and this proof can be found in Section 6. Finally, In Section 7, we discuss an application of our results to the optimization version of the List H-colouring problem for digraphs. Some of the technical proofs omitted from the main body of the paper can be found in Appendices.

2 Basics of approximability

A *combinatorial optimization problem* is defined over a set of *instances* (admissible input data); each instance $\mathcal{I}$ has a finite set $\mathsf{sol}(\mathcal{I})$ of *feasible solutions* associated with it. The *objective function* attributes a positive integer cost to every solution in $\mathsf{sol}(\mathcal{I})$. The goal in an optimization problem is, given an instance $\mathcal{I}$, to find a feasible solution of *optimum* cost. The optimal cost is the largest one for *maximization* problems and the smallest one for *minimization* problems. A combinatorial optimization problem is said to be an **NP** optimization (**NPO**) problem if its instances and solutions can be recognized in polynomial time, the solutions are polynomial-bounded in the input size, and

the objective function can be computed in polynomial time (see, e.g., [2]).

Definition 2.1 (performance ratio) *A solution $s \in \mathsf{sol}(\mathcal{I})$ to an instance $\mathcal{I}$ of an* **NPO** *problem Π is r-approximate if*

$$\max\{\frac{cost(s)}{Opt(\mathcal{I})}, \frac{Opt(\mathcal{I})}{cost(s)}\} \leq r,$$

where $Opt(\mathcal{I})$ is the optimal cost for a solution to $\mathcal{I}$. An approximation algorithm for an **NPO** *problem Π has* performance ratio $\mathcal{R}(n)$ *if, given any instance $\mathcal{I}$ of Π with $|\mathcal{I}| = n$, it outputs an $\mathcal{R}(n)$-approximate solution.*

Definition 2.2 (complexity classes) **PO** *is the class of* **NPO** *problems that can be solved (to optimality) in polynomial time. An* **NPO** *problem Π is in the class* **APX** *if there is a polynomial time approximation algorithm for Π whose performance ratio is bounded by a constant.*

The following result is contained in Proposition 2.3 [11] and its proof.

Lemma 2.3 *Every (weighted or not) problem* Max CSP($\mathcal{F}$) *belongs to* **APX**. *Moreover, if a is the maximum arity of any predicate in $\mathcal{F}$ then there is a polynomial time algorithm which, for every instance $\mathcal{I}$ of* Max CSP($\mathcal{F}$), *produces a solution satisfying at least $\frac{q}{|D|^a}$ constraints, where q is the number of constraints in $\mathcal{I}$.*

Completeness in **APX** is defined using an appropriate reduction, called AP-reduction. Our definition of this reduction follows [13, 31].

Definition 2.4 (AP-reduction, APX-completeness) *An* **NPO** *problem Π_1 is said to be AP-*reducible *to an* **NPO** *problem Π_2 if two polynomial-time computable functions F and G and a constant α exist such that*

(a) for any instance $\mathcal{I}$ of Π_1, $F(\mathcal{I})$ is an instance of Π_2;

(b) for any instance $\mathcal{I}$ of Π_1, and any feasible solution s' of $F(\mathcal{I})$, $G(\mathcal{I}, s')$ is a feasible solution of $\mathcal{I}$;

(c) for any instance $\mathcal{I}$ of Π_1, and any $r \geq 1$, if s' is an r-approximate solution of $F(\mathcal{I})$ then $G(\mathcal{I}, s')$ is an $(1 + (r-1)\alpha + o(1))$-approximate solution of $\mathcal{I}$ where the o-notation is with respect to $|\mathcal{I}|$.

An **NPO** *problem Π is* **APX**-hard *if every problem in* **APX** *is AP-reducible to it. If, in addition, Π is in* **APX** *then Π is called* **APX**-complete.

It is a well-known fact (see, e.g., Section 8.2.1 in [2]) that AP-reductions compose. We shall now give an example of an **APX**-complete problem which will be used extensively in this paper.

Example 2.5 *Given a graph $G = (V, E)$, the* Maximum k-colourable Subgraph *problem, $k \geq 2$, is the problem of maximizing $|E'|$, $E' \subseteq E$, such that the graph $G' = (V, E')$ is k-colourable. This problem is known to be* **APX**-*complete (it is Problem GT33 in [2]). Let neq_k denote the binary disequality predicate on $D = \{0, 1, \ldots, k-1\}$, $k \geq 2$, that is, $neq_k(x, y) = 1 \Leftrightarrow x \neq y$. Consider the problem* Max CSP($\{neq_k\}$) *restricted to instances where every pair of variables appears in the scope of at most one constraint. This problem is exactly the* Maximum k-colourable Subgraph *problem. To see this, think of vertices of a given graph as of variables that take values from D, and*

introduce the constraint $neq_k(x, y)$ for every pair of variables x, y such that (x, y) is an edge in the graph. It follows that the problem Max CSP($\{neq_k\}$) *is* **APX***-complete.*

Note that the weighted Max CSP($\{neq_k\}$) *problem coincides with the well-known problem* Max k-Cut *(it is Problem ND17 in [2]). The* Max 2-Cut *problem is usually referred to as simply* Max Cut.

In some of our hardness proofs, it will be convenient for us to use another type of approximation-preserving reduction, called an L-reduction [2].

Definition 2.6 (L-reduction) *An* **NPO** *problem Π_1 is said to be* L-reducible *to an* **NPO** *problem Π_2 if two polynomial-time computable functions F and G and positive constants α, β exist such that*

(a) given any instance $\mathcal{I}$ of Π_1, algorithm F produces an instance $\mathcal{I}' = F(\mathcal{I})$ of Π_2, such that the cost of an optimal solution for $\mathcal{I}'$, $Opt(\mathcal{I}')$, is at most $\alpha \cdot Opt(\mathcal{I})$;

(b) given $\mathcal{I}, \mathcal{I}' = F(\mathcal{I})$, and any solution s' to $\mathcal{I}'$, algorithm G produces a solution s to $\mathcal{I}$ such that $|cost(s) - Opt(\mathcal{I})| \leq \beta \cdot |cost(s') - Opt(\mathcal{I}')|$.

It is well known (see, e.g., Lemma 8.2 in [2]) that, within **APX**, the existence of an L-reduction from Π_1 to Π_2 implies the existence of an AP-reduction from Π_1 to Π_2.

3 Reduction techniques

The main reduction technique in our **APX**-completeness proofs is based on *strict implementations*, see [13, 31] where this notion was introduced for the Boolean case. We will give this definition in a slightly different form from that of [13, 31], but it can easily be checked to be equivalent to the original one (in the case $|D| = 2$).

Definition 3.1 (strict implementation) *Let $Y = \{y_1, \ldots, y_m\}$ and $Z = \{z_1, \ldots, z_n\}$ be two disjoint sets of variables. The variables in Y are called primary and the variables in Z auxiliary. The set Z may be empty. Let $g_1(\mathbf{y}_1), \ldots, g_s(\mathbf{y}_s)$, $s > 0$, be constraints over $Y \cup Z$. If $g(y_1, \ldots, y_m)$ is a predicate such that the equality*

$$g(y_1, \ldots, y_m) + (\alpha - 1) = \max_Z \sum_{i=1}^{s} g_i(\mathbf{y}_i)$$

holds for all $y_1, \ldots, y_m$, and some fixed $\alpha \in \mathbb{Z}^+$, then this equality is said to be a strict α-implementation *of g from $g_1, \ldots, g_s$.*

We use $\alpha - 1$ rather than α in the above equality to ensure that this notion coincides with the original notion of a strict α-implementation for Boolean constraints [13, 31]. The intuition behind the notion of strict implementation is that it allows one to modify instances while keeping control over costs of solutions. For example, assume that we have a constraint $g(u, v)$ in an instance $\mathcal{I}$ of Max CSP, and there is a strict 2-implementation $g(y_1, y_2) + 1 = \max_z (g_1(y_1, z) + g_2(z, y_2))$. Then the constraint $g(u, v)$ can be replaced by two constraints $g_1(u, z)$, $g_2(z, v)$ such that z does not appear in $\mathcal{I}$, and we know that every solution of cost c to $\mathcal{I}$ can be modified (by choosing an appropriate value for z) to a solution of cost $c + 1$ to the new instance.

We say that a collection of predicates $\mathcal{F}$ *strictly implements* a predicate g if, for some $\alpha \in \mathbb{Z}^+$, there exists a strict α-implementation of g using predicates only from $\mathcal{F}$. In this case we write

$\mathcal{F} \overset{s}{\Longrightarrow}_\alpha f$. We write $\mathcal{F} \overset{s}{\Longrightarrow} f$ if $\mathcal{F} \overset{s}{\Longrightarrow}_\alpha f$ for some α. It is not difficult to show that if f can be obtained from $\mathcal{F}$ by a series of strict implementations then it can also be obtained by a single strict implementation (for the Boolean case, this is shown in Lemma 5.8 [13]). In this paper, we will use about 60 specific strict implementations for the case when $|D| = 4$. Each of them can be straightforwardly verified by hand, or by a simple computer program[1].

The following lemma is a simple (but important) example of how strict implementations work.

Lemma 3.2 *$\mathcal{C}_D$ strictly implements every predicate in $\mathcal{U}_D$.*

Proof: It is easy to see that, for any $D' \subseteq D$, $u_{D'}(x) = \sum_{d \in D'} u_{\{d\}}(x)$ is a strict 1-implementation. □

In our proofs, we will use problems with the bounded occurrence property, and we now introduce notation for such problems.

Definition 3.3 (bounded occurrence problems) Max CSP($\mathcal{F}$) $- k$ *will denote the problem* Max CSP($\mathcal{F}$) *restricted to instances with the number of occurrences of variables is bounded by k. We will write that* Max CSP($\mathcal{F}$) $- B$ *is* **APX***-complete to denote that* Max CSP($\mathcal{F}$) $- k$ *is* **APX***-complete for some k.*

Note that, by definition, repetitions of constraints in instances of Max CSP are allowed. If a variable occurs t times in a constraint which appears s times in an instance, then this would contribute $t \cdot s$ to the number of occurrences of that variable in the instance.

Lemma 3.4 *If $\mathcal{F}$ strictly implements a predicate f, and* Max CSP($\mathcal{F} \cup \{f\}$)$-B$ *is* **APX***-complete, then* Max CSP($\mathcal{F}$) $- B$ *is* **APX***-complete as well.*

Proof: This lemma for the Boolean case, but without the assumption on bounded occurrences, is Lemma 5.18 in [13]. Our proof is almost identical to the proof of Lemma 5.18 in [13], and it uses the same AP-reduction. Essentially, we only need to verify that the mapping F in this reduction preserves the bounded occurrence property.

Let k be a number such that Max CSP($\mathcal{F} \cup \{f\}$) $- k$ is **APX**-complete and let $\alpha \in \mathbb{Z}^+$ be such that $\mathcal{F} \overset{s}{\Longrightarrow}_\alpha f$. Take an arbitrary instance $\mathcal{I}$ of Max CSP($\mathcal{F} \cup \{f\}$) $- k$. Note that every predicate in $\mathcal{F}$ can be (trivially) strictly α-implemented from $\mathcal{F}$ in such a way that each auxiliary variable appears only once in the strict implementation (simply use any satisfiable collection of $\alpha - 1$ constraints with no repetitions of variables); this is a small technicality which ensures uniformity in the following transformation of instances. Replace every constraint in $\mathcal{I}$ by a set of constraints appearing in the right-hand side of its strict α-implementation from $\mathcal{F}$, keeping the same primary variables and using fresh copies of auxiliary variables every time. Denote the obtained instance by $\mathcal{I}'$. The function F in this AP-reduction will be such that $F(\mathcal{I}) = \mathcal{I}'$ for all $\mathcal{I}$. Let t be the maximum number of occurrences of a variable (primary or auxiliary) in the right-hand side of the strict implementation of f from $\mathcal{F}$. It is clear that $\mathcal{I}'$ is an instance of Max CSP($\mathcal{F}$), and that the number of occurrences of any variable in $\mathcal{I}'$ is bounded by $k' = tk$.

Let V' be the set of variables in $\mathcal{I}'$. Let $\varphi' : V' \to D$ be an r-approximate solution to $\mathcal{I}'$. The mapping G uses two possible solutions to $\mathcal{I}$ and takes the better of the two. The first solution is

[1] An example of such a program can be obtained from the authors or be anonymously downloaded from `http://www.ida.liu.se/~petej/supermodular.html`.

$\varphi'|_V$, while the second is a solution satisfying $\beta = \frac{q}{|D|^a}$ constraints which exists by Lemma 2.3 (here a is the maximum arity of constraints in $\mathcal{F} \cup \{f\}$).

One can show, by literally repeating the argument in the proof of Lemma 5.18 in [13], that $G(\varphi')$ is an r'-approximate solution to $\mathcal{I}$ where $r' \leq 1 + \gamma(r-1)$ with $\gamma = \beta(\alpha - 1) + 1$.

We have constructed an AP-reduction from Max CSP$(\mathcal{F} \cup \{f\}) - k$ to Max CSP$(\mathcal{F}) - k'$, thus proving the lemma. □

Lemma 3.4 will be used as follows in our **APX**-completeness proofs: if $\mathcal{F}'$ is a fixed finite collection of predicates each of which can be strictly implemented by $\mathcal{F}$ then we can assume that $\mathcal{F}' \subseteq \mathcal{F}$. For example, if $\mathcal{F}$ contains a binary predicate f then we can assume, at any time when it is convenient, that $\mathcal{F}$ also contains $f'(x, y) = f(y, x)$, since this equality is a strict 1-implementation of f'.

Finally, we will use a technique based on domain restriction. For a subset $D' \subseteq D$, let $\mathcal{F}|_{D'} = \{f|_{D'} \mid f \in \mathcal{F} \text{ and } f|_{D'} \text{ is non-trivial}\}$.

Lemma 3.5 *Let $D' \subseteq D$ and $u_{D'} \in \mathcal{F}$. If* Max CSP$(\mathcal{F}|_{D'}) - B$ *is* **APX***-complete then so is* Max CSP$(\mathcal{F}) - B$.

Proof: Let k be a bound on the number of occurences such that Max CSP$(\mathcal{F}|_{D'}) - k$ is **APX**-complete. We establish an L-reduction from Max CSP$(\mathcal{F}|_{D'}) - k$ to Max CSP$(\mathcal{F}) - k'$ where $k' = 2k$.

An instance $\mathcal{I}$ of Max CSP$(\mathcal{F}|_{D'}) - k$ corresponding to $f(x_1, \ldots, x_n) = \sum_{i=1}^{q} f_i(\mathbf{x}_i)$ will be mapped to an instance $\mathcal{I}'$ corresponding to $f'(x_1, \ldots, x_n) = \sum_{i=1}^{q} f'_i(\mathbf{x}_i) + k \sum_{i=1}^{n} u_{D'}(x_i)$ where each $f'_i \in \mathcal{F}$ is such that $f'_i|_{D'} = f_i$. We may without loss of generality assume that all n variables x_i actually appear in constraint scopes in $\mathcal{I}$. Note that $\mathcal{I}'$ is indeed an instance of Max CSP$(\mathcal{F}) - k'$.

Let $V = \{x_1, \ldots, x_n\}$ and fix an element $d \in D'$. If $\varphi' : V \to D$ is a solution to $\mathcal{I}'$, then it is modified to a solution to $\mathcal{I}$ as follows: set $\varphi(x_i) = d$ whenever $\varphi'(x_i) \notin D'$, and $\varphi(x_i) = \varphi'(x_i)$ otherwise.

We will show that this pair of mappings is an L-reduction for suitable α and β.

Note that, for any solution to $\mathcal{I}'$, changing all values outside of D' to any values in D' can only increase the cost of the solution. This follows from the fact that, by changing any value outside of D' to a value in D', we can lose at most k satisfied constraints, but we satisfy k constraints of the form $u_{D'}(x)$. It follows that $Opt(\mathcal{I}') = Opt(\mathcal{I}) + kn$.

Let a be the maximum arity of constraints in $\mathcal{F}|_{D'}$. Let $c = \frac{1}{|D|^a}$. Then we have $c \cdot q \leq Opt(\mathcal{I})$ by Lemma 2.3 (recall that q is the number of constraints in $\mathcal{I}$). Set $\alpha = \frac{ak}{c} + 1$. Note that we have $n \leq aq$ because the total length of constraint scopes in $\mathcal{I}$ is at least n and at most aq. Since $n \leq aq \leq \frac{aOpt(\mathcal{I})}{c}$, we have

$$Opt(\mathcal{I}') = Opt(\mathcal{I}) + kn \leq Opt(\mathcal{I}) + k\frac{aOpt(\mathcal{I})}{c} = \alpha \cdot Opt(\mathcal{I}),$$

so the first property of an L-reduction is satisfied.

We will now show that the second property is satisfied with $\beta = 1$. Let φ' and φ be solutions to $\mathcal{I}'$ and $\mathcal{I}$, respectively, such as described above.

Let V_1 be the set of variables which φ' sends to $D \setminus D'$, and V_2 the variables sent to D'; set $r = |V_2|$. Divide all constraints in $\mathcal{I}'$ into three pairwise disjoint groups: C_1 consists of all constraints $f_i(\mathbf{x}_i)$ that contain at least one variable from V_1, C_2 of all constraints $f_i(\mathbf{x}_i)$ that use variables only from V_2, and C_3 contains the kn constraints of the form $u_{D'}(x_i)$. Let $q_1 = |C_1|$. Furthermore,

let s_1 and s_2 be the numbers of constraints in C_1 and C_2, respectively, that are satisfied by φ'. By the bounded occurrence property, we have $s_1 \leq q_1 \leq (n-r)k$. In particular, it follows that $s_1 - nk + rk \leq 0$. Note also that $cost(\varphi') = s_1 + s_2 + rk$ and $s_2 \leq cost(\varphi)$. Finally, we have

$$Opt(\mathcal{I}) - cost(\varphi) \leq Opt(\mathcal{I}) - s_2 =$$

$$[Opt(\mathcal{I}) + nk] - [s_1 + s_2 + rk] + [s_1 - nk + rk] \leq Opt(\mathcal{I}') - cost(\varphi').$$

□

4 Supermodularity, Monge properties, and Max CSP

4.1 Basics of supermodularity

In this section we discuss the well-known combinatorial algebraic property of supermodularity [40] which will play a crucial role in classifying the approximability of Max CSP problems.

A partial order on a set D is called a *lattice order* if, for every $x, y \in D$, there exists a greatest lower bound $x \sqcap y$ and a least upper bound $x \sqcup y$. The corresponding algebra $\mathcal{L} = (D, \sqcap, \sqcup)$ is called a *lattice*. For tuples $\mathbf{a} = (a_1, \ldots, a_n)$, $\mathbf{b} = (b_1, \ldots, b_n)$ in D^n, let $\mathbf{a} \sqcap \mathbf{b}$ and $\mathbf{a} \sqcup \mathbf{b}$ denote the tuples $(a_1 \sqcap b_1, \ldots, a_n \sqcap b_n)$ and $(a_1 \sqcup b_1, \ldots, a_n \sqcup b_n)$, respectively.

Definition 4.1 (supermodular function) *Let $\mathcal{L}$ be a lattice on D. A function $f : D^n \to \mathbb{Z}^+$ is called* supermodular *on $\mathcal{L}$ if*

$$f(\mathbf{a}) + f(\mathbf{b}) \leq f(\mathbf{a} \sqcap \mathbf{b}) + f(\mathbf{a} \sqcup \mathbf{b}) \text{ for all } \mathbf{a}, \mathbf{b} \in D^n.$$

Note that predicates are functions, so it makes sense to consider supermodular predicates. We say that $\mathcal{F} \subseteq R_D$ is supermodular on $\mathcal{L}$ if every $f \in \mathcal{F}$ has this property.

A finite lattice $\mathcal{L} = (D, \sqcap, \sqcup)$ is *distributive* if and only if it can be represented by subsets of a set A, where the operations $\sqcap$ and $\sqcup$ are interpreted as set-theoretic intersection and union, respectively. Totally ordered lattices, or *chains*, will be of special interest in this paper. Note that, for chains, the operations $\sqcap$ and $\sqcup$ are simply min and max. Hence, the supermodularity property for an n-ary function f on a chain is expressed as follows:

$$f(a_1, \ldots, a_n) + f(b_1, \ldots, b_n) \leq$$

$$f(\min(a_1, b_1), \ldots, \min(a_n, b_n)) + f(\max(a_1, b_1), \ldots, \max(a_1, b_1))$$

for all $a_1, \ldots, a_n, b_1, \ldots, b_n$.

Example 4.2

1) The disequality predicate neq_D is not supermodular on any chain on D. Take two elements $d_1, d_2 \in D$ such that $d_1 < d_2$. Then

$$neq_D(d_1, d_2) + neq_D(d_2, d_1) = 2 \not\leq 0 = neq_D(d_1, d_1) + neq_D(d_2, d_2).$$

2) Fix a chain on D and let $\mathbf{a}, \mathbf{b}$ be arbitrary elements of D^2. Consider the binary predicate $f_{\mathbf{a}}$, $f^{\mathbf{b}}$ and $f^{\mathbf{b}}_{\mathbf{a}}$ defined by the rules

$$\begin{aligned} f_{\mathbf{a}}(x, y) = 1 &\quad \Leftrightarrow \quad (x, y) \leq \mathbf{a}, \\ f^{\mathbf{b}}(x, y) = 1 &\quad \Leftrightarrow \quad (x, y) \geq \mathbf{b}, \\ f^{\mathbf{b}}_{\mathbf{a}}(x, y) = 1 &\quad \Leftrightarrow \quad (x, y) \leq \mathbf{a} \text{ or } (x, y) \geq \mathbf{b}, \end{aligned}$$

where the order on D^2 is component-wise. It is easy to check that every predicate defined above in this part of the example is supermodular on the chain. Note that such predicates were considered in [11] where they were called generalized 2-monotone. We will see later in this subsection (Lemma 4.4) that such predicates are generic supermodular binary predicates on a chain.

We will now make some simple, but useful, observations.

Observation 4.3

1. *Any chain is a distributive lattice.*
2. *Any unary predicate on D is supermodular on any chain on D.*
3. *A predicate is supermodular on a chain if and only if it is supermodular on its dual chain (obtained by reversing the order).*

Given a chain in D, any binary function f on D can be represented as a $|D| \times |D|$ matrix M such that $M(x, y) = f(x, y)$; here the chain indicates the order of indices of M, and $M(x, y)$ is the entry in row x and column y of M. Note that this matrix is essentially the table of values of the predicate. For example, some binary predicates on $D = \{0, 1, 2, 3\}$ that are supermodular on the chain $0 < 1 < 2 < 3$ are listed in Fig. 1 (these predicates will be used later in the proof of Theorem 6.3). Note that all predicates in Fig. 1 have the form described in Example 4.2(2). For example, h_2 is $f^{(3,3)}_{(0,1)}$ and h_{17} is $f^{(1,3)}_{(2,1)}$.

h_1
1000
0000
0000
0001

h_2
1100
0000
0000
0001

h_3
1110
0000
0000
0001

h_4
1100
1100
0000
0001

h_5
1110
1110
0000
0001

h_6
1110
1110
1110
0001

h_7
1100
0000
0001
0001

h_8
1110
0000
0001
0001

h_9
1000
1000
0001
0001

h_{10}
1100
1100
0001
0001

h_{11}
1110
1110
0001
0001

h_{12}
1110
0001
0001
0001

h_{13}
1000
1001
0001
0001

h_{14}
1100
1101
0001
0001

h_{15}
1000
1001
1001
0001

h_{16}
1100
1100
1101
0001

h_{17}
1100
1101
1101
0001

h_{18}
1100
1100
0011
0011

Figure 1: A list of predicates on $\{0, 1, 2, 3\}$ which are supermodular on the chain $0 < 1 < 2 < 3$. The predicates are represented by tables of values.

A square matrix M is called *anti-Monge* (or a-Monge, for short)[2] if $M(i, s) + M(r, j) \leq M(i, j) + M(r, s)$ for all $i < r$ and $j < s$. It is well known (and easy to check) that matrices corresponding to binary supermodular functions on a chain are precisely the a-Monge matrices (see, e.g., Observation 6.1 in [10]). Hence, one can view the tables in Fig. 1 as a-Monge matrices. We will be particularly interested in binary supermodular *predicates* on chains, and the next result describes the structure of 0-1 a-Monge square matrices.

In order to make the correspondence between matrices and binary functions more transparent, we will use the set $J = \{0, \ldots, n-1\}$ to number rows and columns of an $n \times n$ matrix. Let L^{pq}_n denote the square 0-1 matrix of size n such that $L^{pq}_n(i, j) = 1$ if and only if $i \leq p$ and $j \leq q$. Similarly, R^{st}_n denotes the square 0-1 matrix of size n such that $R^{st}_n(i, j) = 1$ if and only if $i \geq s$ and $j \geq t$. Let U and W be two subsets of J. We denote by $M[U, W]$ the $|U| \times |W|$ submatrix of M that is obtained by deleting all rows not contained in U and all columns not in W. Expression $M[U, W] = a$ will mean that all elements in the submatrix are equal to a.

The following result is a direct corollary of Lemma 2.3 of [10].

[2]Other names used for such matrices are *inverse Monge* and *dual Monge*.

Lemma 4.4 *A non-zero 0-1 matrix M of size $n \times n$ without all-ones rows and columns is an a-Monge matrix if and only if one of the following holds*

- $M = L_n^{pq}$, *for some* $0 \leq p, q \leq n-2$, *or*
- $M = R^{st}$, *for some* $1 \leq s, t \leq n-1$, *or*
- $M = L_n^{pq} + R_n^{st}$ *for some* $0 \leq p, q \leq n-2$ *and* $1 \leq s, t \leq n-1$, *with* $p < s$, *or* $q < t$, *or both.*

The family of n-ary supermodular functions on a chain was also studied under the name of n-dimensional *anti-Monge arrays* [10]. As a special case of Lemma 6.3 of [10], we have the following result (see also Observation 6.1 of [10]).

Lemma 4.5 *An n-ary, $n \geq 2$, function f is supermodular on a fixed chain if and only if the following holds: every binary function obtained from f by replacing any given $n-2$ variables by any constants is supermodular on this chain.*

4.2 Supermodularity and Max CSP

The property of supermodularity has been used to classify the approximability of problems Max CSP($\mathcal{F}$) for small sets D (though, originally the classification for the case $|D| = 2$ was obtained and stated in [12, 13, 31] without using this property). To make use of results in [11, 28], we need to introduce some more notation.

Definition 4.6 (endomorphism, core) *An* endomorphism *of $\mathcal{F}$ is a unary operation μ on D such that, for all $f \in \mathcal{F}$ and all $(a_1, \ldots, a_m) \in D^m$, we have*

$$f(a_1, \ldots, a_m) = 1 \Rightarrow f(\mu(a_1), \ldots, \mu(a_m)) = 1.$$

We will say that $\mathcal{F}$ is a core *if every endomorphism of $\mathcal{F}$ is injective (i.e., a permutation).*

If μ is an endomorphism of $\mathcal{F}$ with a minimal image $im(\mu) = D'$ then a core of $\mathcal{F}$, denoted $core(\mathcal{F})$, is the set $\mathcal{F}|_{D'}$.

The intuition here is that if $\mathcal{F}$ is not a core then it has a non-injective endomorphism μ, which implies that, for every assignment φ, there is another assignment $\mu\varphi$ that satisfies all constraints satisfied by φ and uses only a restricted set of values, so the problem is equivalent to a problem over this smaller set. As in the case of graphs, all cores of $\mathcal{F}$ are isomorphic, so one can speak about *the* core of $\mathcal{F}$. Note that any set of the form $\mathcal{F} \cup \mathcal{C}_D$ is a core.

Theorem 4.7 ([11, 13, 28]) *Let $|D| \leq 3$ and let $\mathcal{F} \subseteq R_D$ be a core. If $\mathcal{F}$ is supermodular on some chain on D then weighted* Max CSP($\mathcal{F}$) *belongs to* **PO**. *Otherwise,* Max CSP($\mathcal{F}$) *is* **APX**-*complete.*

Remark 4.8 *It was shown in Lemma 5.37 of [13] that, for $D = \{0, 1\}$, $\mathcal{F} \subseteq R_{\{0,1\}}$ can strictly implement neq_2 whenever* Max CSP($\mathcal{F}$) *is* **APX**-*complete in the above theorem (i.e. whenever $\mathcal{F}$ is a core that is not supermodular on any chain). Moreover, it follows from (the proof of) Theorem 3 of [28] that if $|D| = 3$ and $\mathcal{F}$ is not supermodular on any chain on D then $\mathcal{F} \cup \mathcal{C}_D$ can express neq_2 or neq_3 by using a sequence of the following operations:*

- *adding to $\mathcal{F}$ a predicate that can be strictly implemented from $\mathcal{F}$*
- *taking the core of a subset of $\mathcal{F}$.*

It was shown in [1] that MAX CUT remains **APX**-complete even when restricted to cubic graphs. Since MAX CUT is the same problem as MAX CSP($\{neq_2\}$) (see Example 2.5), it follows that MAX CSP($\{neq_2\}$) $- B$ is **APX**-complete. Moreover, since $neq_k|_{\{0,1\}} = neq_2$, it follows from Lemma 3.5 that MAX CSP($\{neq_k, u_{\{0,1\}}\}$) $- B$ is **APX**-complete for any k. Therefore, we obtain the following corollary by combining Remark 4.8 with Lemmas 3.4 and 3.5.

Corollary 4.9 *Let $|D| \leq 3$ and $\mathcal{F}$ not supermodular on any chain on D. Then the problem* MAX CSP($\mathcal{F} \cup \mathcal{U}_D$) $- B$ *is* **APX***-complete.*

The tractability part of our classification is contained in the following result:

Theorem 4.10 ([11], see also [26, 39]) *If $\mathcal{F}$ is supermodular on some distributive lattice on D, then weighted* MAX CSP($\mathcal{F}$) *is in* **PO**.

5 Permuted a-Monge matrices

In this section, we prove results about a-Monge matrices that will imply, via the correspondence between binary supermodular predicates on chains and a-Monge matrices, the following result.

Theorem 5.1 *If $\mathcal{F}$ is a set of binary predicates that is not supermodular on any chain on D, then there exists $\mathcal{F}' \subseteq \mathcal{F}$ with $|\mathcal{F}'| \leq 3$ and $D' \subseteq D$ with $|D'| \leq 4$ such that $\mathcal{F}'|_{D'}$ is not supermodular on any chain on D'.*

We prove this theorem in two steps: the existence of D' is established in Section 5.1 (see Corollary 5.4) and the existence of $\mathcal{F}'$ in Section 5.2 (see Proposition 5.7). Our results about matrices will be more general than required to prove Theorem 5.1 because we will consider general (i.e., not necessarily 0-1) matrices.

First, we need to introduce some concepts and notation. Let M be an $n \times n$ matrix. If there is a permutation π that *simultaneously* permutes rows and columns of M so that the resulting matrix is an a-Monge matrix, then the matrix M is called a *permuted* a-Monge matrix and the permutation is called an *a-Monge permutation* for M. Note that we will often use the term 'permutation' as a synonym for 'linear (re-)ordering'. Given a set of indices $I = \{i_1, \ldots, i_k\} \subseteq J = \{0, \ldots, n-1\}$, we use notation $M[I]$ for the sub-matrix $M[I, I]$. We say that $M[I]$ is permuted according to a permutation $\langle s_1, \ldots, s_k \rangle$, where $I = \{s_1, \ldots, s_k\}$, if row (column) s_1 is the first row (column) in the permuted matrix, s_2 is the second row (column), and so on. If $n \leq 4$ and M is not a permuted a-Monge matrix, then M is called a *bad* matrix.

A row i *precedes* a row j in M ($i \prec j$ for short), if row i occurs before row j in M. If i precedes j in a permutation π, then we write $i \prec_\pi j$. When the permutation π is understood from the context, we simply write $i \prec j$. If π is an a-Monge permutation for the matrix M, then the *reverse* of π, π^- defined as $\pi^-(i) = \pi(n-1-i)$, is also an a-Monge permutation. Therefore, given two indices i and j, we can always assume that i precedes j in an a-Monge permutation (if there is any).

Denote by $\Delta(i, j, k, l)$, for $i, j, k, l \in J$, an algebraic sum that involves four entries of the matrix M: $\Delta(i, j, k, l) = M(i, k) + M(j, l) - M(i, l) - M(j, k)$. Given a permutation π for permuting rows and columns in M, we use a similar notation for the sums in the permuted matrix: $\Delta(i, j, k, l, \pi) = M(\pi(i), \pi(k)) + M(\pi(j), \pi(l)) - M(\pi(i), \pi(l)) - M(\pi(j), \pi(k))$. For $k = i$ and $l = j$, we use simplified notation $\Delta(i, j) = \Delta(i, j, i, j)$, for $i, j \in J$. Matrix M is an a-Monge matrix if and only

if $\Delta(i,j,k,l) \geq 0$ for all $i < j$ and $k < l$, and M is permuted a-Monge if and only if there exists a permutation π such that $\Delta(i,j,k,l,\pi) \geq 0$ for all $i < j$ and $k < l$. It is easy to check that

$$\Delta(i,j,k,l) = \sum_{s=i,\ldots,j-1;t=k,\ldots,l-1} \Delta(s,s+1,t,t+1). \quad (1)$$

Therefore, given a permutation π and matrix M, it can be checked in $O(n^2)$ time whether π is an a-Monge permutation for the matrix M.

We will often use the following equalities (which are direct consequences of equation (1)):

$$\Delta(i,j,k,l) = \Delta(i,s,k,l) + \Delta(s,j,k,l) \text{ and } \Delta(i,j,k,l) = \Delta(i,j,k,s) + \Delta(i,j,s,l)$$

We will say that row (column) s is equivalent to row (respectively, column) t, if $\Delta(s,t,k,l) = 0$ (respectively, $\Delta(k,l,s,t) = 0$) for all k,l. It can easily be shown that if rows s and t are equivalent, then $M(s,i) = M(t,i) + \alpha_{st}$, for all i and some constant α_{st}. Hence, after subtracting α_{st} from all elements in the row s ($M'(s,i) = M(s,i) - \alpha_{st}$), one gets two identical rows s and t ($M'(s,i) = M'(t,i)$ for all i). A matrix with all rows (and all columns) equivalent is called a sum matrix: It can be shown that in this case $M(s,t) = u_s + v_t$, for some real vectors u and v. Clearly, any sum matrix is a-Monge.

5.1 Reducing the size of matrices

We will first show that whenever an $n \times n$ matrix M is not a permuted a-Monge matrix, then there exists a set of indices B with $|B| \leq 4$ such that $M[B]$ is a bad matrix. Our approach to the recognition of bad matrices is loosely based on the COM (Construct partial Orders and Merge them) algorithm, suggested in [15]. This algorithm constitutes a general approach to deciding whether a given matrix, possibly with some unknown elements, can be permuted to avoid a special set of 2×2 submatrices. In our case, these are submatrices $M[\{i,j,k,l\}]$ with $\Delta(i,j,k,l) < 0$.

We will use the idea of the COM algorithm, which goes as follows: given a matrix M, we try to construct an a-Monge permutation for it. We start with a pair of indices (i,j) which correspond to two non-equivalent rows or columns in the matrix. We assume further that the index i precedes index j in an a-Monge permutation π. The assumption $i \prec_\pi j$ determines the order of some other indices. Under the assumption that $i \prec_\pi j$, the strict inequality $\Delta(i,j,k,l) > 0$ indicates that $k \prec_\pi l$, while the strict inequality $\Delta(i,j,k,l) < 0$ indicates that $l \prec_\pi k$. (Note that $\Delta(i,j,k,l) = -\Delta(i,j,l,k) = -\Delta(j,i,k,l) = \Delta(j,i,l,k)$ – this property will often be used in our proofs). The obtained information can be conveniently represented as a directed graph P_M with nodes J and directed arcs corresponding to the identified precedence constraints together with the initial constraint $i \prec_\pi j$. We then extend P_M recursively to obtain additional information about the ordering of indices. Eventually, either P_M contains an oriented cycle which signals that the matrix is not a permuted a-Monge matrix, or we can view P_M as a partial order. This order defines a set of permutations (i.e., linear extensions of P_M) which are our candidates for an a-Monge permutation. We illustrate the COM approach with the following example.

Example 5.2 Consider a submatrix $M[\{i,k,j\}]$. Schematic representation of this sub-matrix and algebraic sums Δ is shown in Fig. 2. We claim that, provided $\Delta(i,j) = \Delta(i,k) = \Delta(k,j) = 0$, the submatrix is either a bad matrix or a sum matrix.

It follows from $\Delta(i,j) = \Delta(i,k) + \Delta(i,k,k,j) + \Delta(k,j,i,k) + \Delta(k,j) = 0$ that $\Delta(k,j,i,k) = -\Delta(i,k,k,j)$. Suppose that $\Delta(k,j,i,k) \neq 0$ and $\Delta(i,k,k,j) \neq 0$. Without loss of generality, suppose that $i \prec k$ in an a-Monge permutation and that $\Delta(i,k,k,j) > 0$. The assumption that row

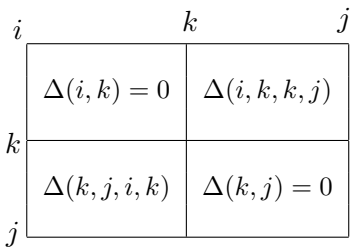

Figure 2: Schematic representation of submatrices and algebraic sums Δ.

i precedes row k, together with the inequality $\Delta(i,k,k,j) > 0$ yields $k \prec j$. The assumption that column i precedes column k together with the inequality $\Delta(k,j,i,k) < 0$ yields a contradictory precedence $j \prec k$. Therefore $M[\{i,j,k\}]$ is a bad matrix.

If $\Delta(k,j,i,k) = 0$ and $\Delta(i,k,k,j) = 0$, then it can easily be shown that $M[\{i,j,k\}]$ is a sum matrix. □

We recommend the reader to use diagrams like the one in Fig. 2 in the following proof, since they make arguments more transparent.

Theorem 5.3 *If an $n \times n$ matrix M is not a permuted a-Monge matrix, then there exists a set of indices B with $|B| \leq 4$, such that $M[B]$ is a bad matrix.*

Proof: We can without loss of generality assume that M has no pair s,t of indices such that both rows s,t are equivalent and columns s,t are equivalent. Indeed, if s,t is such a pair then it is easy to see that M is permuted a-Monge if and only if $M[J \setminus \{s\}]$ is permuted a-Monge, so we can delete row s and column s and continue.

First note that if there exists a pair i,j such that $i \neq j$ and $\Delta(i,j) < 0$, then $M[\{i,j\}]$ is a bad matrix, so we assume further on that $\Delta(i,j) \geq 0$ for all distinct i,j.

Assume that $\Delta(i,j) = 0$ for all i,j. Suppose that there exists a triple i,j,k such that $\Delta(k,j,i,k) \neq 0$ and/or $\Delta(i,k,k,j) \neq 0$. Then, as shown in the example above, $M[\{i,j,k\}]$ is a bad matrix in this case. Suppose instead that $\Delta(k,j,i,k) = 0$ and $\Delta(i,k,k,j) = 0$ for all i,k,j. For any s,t,k,l with $s,t < k,l$, we have $\Delta(s,t,k,l) = \Delta(s,t,t,l) - \Delta(s,t,t,k)$, and therefore $\Delta(s,t,k,l) = 0$. For any s,t,k,l with $s,t > k,l$, we have $\Delta(s,t,k,l) = \Delta(s,t,k,s) - \Delta(s,t,l,s)$, and therefore $\Delta(s,t,k,l) = 0$. It can be shown in a similar way that $\Delta(s,t,k,l) = 0$ for all s,t,k,l, and therefore M is a sum matrix, which is impossible because M is not permuted a-Monge.

Assume now that $\max_{k,l} \Delta(k,l) > 0$. We will try to construct an a-Monge permutation for M, and show that such an effort unavoidably results in the identification of a bad submatrix in M. If M were a permuted a-Monge matrix, then there would exist indices $i^\star, j^\star$ and an (a-Monge) permutation π with $\pi(i^\star) = 0$ and $\pi(j^\star) = n-1$ such that $\Delta(i^\star, j^\star) = \max_{k,l} \Delta(k,l)$ (see equation (1)), and also $\Delta(i^\star, j^\star, j, j+1, \pi) \geq 0$ and $\Delta(j, j+1, i^\star, j^\star, \pi) \geq 0$ for all $j = 0, 1, \ldots, n-2$. To simplify the presentation, we assume that $i^\star = 0$ and $j^\star = n-1$ (otherwise, we renumber the rows and columns in the matrix). The above inequalities can be rewritten as $M(0,\pi(j)) - M(n-1,\pi(j)) \geq M(0,\pi(j+1)) - M(n-1,\pi(j+1))$ and $M(\pi(j),0) - M(\pi(j),n-1) \geq M(\pi(j+1),0) - M(\pi(j+1),n-1)$ for $j = 0,1,\ldots,n-2$.

So, an a-Monge permutation π would have to sort the differences $(M(0,i) - M(n-1,i))$ and the differences $(M(i,0) - M(i,n-1))$, $i \neq 0, n-1$ in non-increasing order. If there exists no permutation that sorts both sequences, then there is a pair i,j such that $M(0,i) - M(n-1,i) < M(0,j) - M(n-$

$1, j)$ (which yields the precedence constraint $i \prec j$) and $M(i,0)-M(i,n-1) > M(j,0)-M(j,n-1)$ (which yields the precedence constraint $j \prec i$). This implies that matrix $M[\{0, n-1, i, j\}]$ is a bad matrix.

Suppose now that $\Delta(0, n-1) = \max_{k,l} \Delta(k,l)$ and there exists a permutation π, with $\pi(0) = 0$ and $\pi(n-1) = n-1$, that sorts both sequences. Fix such a permutation and permute M according to it. We can without loss of generality assume that M had this new form from the very beginning, that is, both the sequence $(M(0,i)-M(n-1,i))$ and the sequence $(M(i,0)-M(i,n-1))$, $i \neq 0, n-1$, are already in non-increasing order.

Since M is not permuted a-Monge, we still have indices p, q, s, t such that $p < q$, $s < t$, and $\Delta(p,q,s,t) < 0$. It follows from the inequality $\Delta(p,q,s,t) < 0$ and from the equation (1) that there exists an index i with $p \leq i \leq q-1$, and an index k with $s \leq k \leq t-1$, such that $\Delta(i, i+1, k, k+1) < 0$. We consider the case $i < k$: the case $i = k$ is already eliminated and the case $i > k$ is symmetric.

Assume that there exist indices i and k with $i+1 < k$ such that $\Delta(i, i+1, k, k+1) < 0$, $\Delta(i, i+1, i+1, k) > 0$, and $\Delta(i+1, k, k, k+1) > 0$. We claim that $M[\{i, i+1, k, k+1\}]$ is a bad matrix in this case. Indeed, the assumption $i \prec i+1$ yields $i+1 \prec k$ and $k+1 \prec k$, and constraint $k+1 \prec k$ for the columns k and $k+1$ yields $k \prec i+1$. This proves the claim.

Assume now that there is *no* pair of indices i, k with $i+1 < k$ such that $\Delta(i, i+1, k, k+1) < 0$, $\Delta(i, i+1, i+1, k) > 0$, and $\Delta(i+1, k, k, k+1) > 0$. We claim that then there exists a triple of indices i, j, l with $i < j < l$ such that $\Delta(i, j, j, l) < 0$. Indeed, we know that we have a pair of indices i, k with $i < k$ such that $\Delta(i, i+1, k, k+1) < 0$. If $i+1 = k$, then our claim trivially holds with $j = k$ and $l = k+1$. Otherwise, we have $i+1 < k$ with $\Delta(i, i+1, i+1, k) \leq 0$ or $\Delta(i+1, k, k, k+1) \leq 0$ (or both). If $\Delta(i, i+1, i+1, k) \leq 0$ then $\Delta(i, i+1, i+1, k+1) = \Delta(i, i+1, i+1, k) + \Delta(i, i+1, k, k+1) < 0$, so we can take $j = i+1$ and $l = k+1$ in our claim. The situation when $\Delta(i+1, k, k, k+1) \leq 0$ is treated similarly.

We consider two cases:

> Case 1 There exists a triple $i < j < l$ with $\Delta(i, j, j, l) < 0$ such that $i = 0$ or $l = n-1$, or both.
>
> We consider the case with $\Delta(0, j, j, l) < 0$ (the case of $\Delta(i, j, j, n-1) < 0$ is symmetric). We claim that matrix $M[\{0, j, l, n-1\}]$ is a bad matrix in this case. Indeed, rows and columns in the matrix are sorted to guarantee, in particular, the inequalities $\Delta(0, n-1, j, l) \geq 0$ and $\Delta(0, j, 0, n-1) \geq 0$. It follows from the assumption $\Delta(0, j, j, l) < 0$ and the equality $\Delta(0, n-1, j, l) = \Delta(0, j, j, l) + \Delta(j, n-1, j, l)$ that $\Delta(j, n-1, j, l) > 0$. So, the assumption $0 \prec j$ yields $l \prec j$, and $l \prec j$ yields $n-1 \prec j$. We will show that then we have a contradiction with the choice of 0 and $n-1$ as a pair such that $\Delta(0, n-1) = \max_{k,l} \Delta(k,l)$.
>
> If $l = n-1$ then there are two permutations of $\{0, j, n-1\}$ compatible with the obtained precedence constraints: $\langle 0, n-1, j\rangle$ and $\langle n-1, 0, j\rangle$. If $\langle 0, n-1, j\rangle$ is an a-Monge permutation for $M[\{0, j, n-1\}]$ then $\Delta(0, j)$ can be represented as a sum of non-negative numbers (see equality (1)) which include, in particular, $\Delta(0, n-1)$ and $\Delta(n-1, j) > 0$. This is a contradiction with the choice of 0 and $n-1$. If $\langle n-1, 0, j\rangle$ is an a-Monge permutation for $M[\{0, j, n-1\}]$ then we get a contradiction in a similar way, using the fact that $\Delta(0, j) = \Delta(0, j, 0, n-1) - \Delta(0, j, j, n-1) > 0$.
>
> Assume now $l \neq n-1$. This means that $\Delta(0, j, j, n-1) \geq 0$. Moreover, since $\Delta(0, j, j, l) < 0$ and $\Delta(0, j, j, n-1) = \Delta(0, j, j, l) + \Delta(0, j, l, n-1)$, we also have $\Delta(0, j, l, n-1) > 0$. In addition to the previously stated precedence constraints $0 \prec j$, $l \prec j$, and $n-1 \prec j$, the last inequality implies a new constraint $l \prec n-1$. So we have three possible permutations for

permuting the submatrix $M[\{0, j, l, n-1\}]$ to an a-Monge matrix: $\langle 0, l, n-1, j\rangle$, $\langle l, 0, n-1, j\rangle$, and $\langle l, n-1, 0, j\rangle$.

Assume that $\langle 0, l, n-1, j\rangle$ is an a-Monge permutation. Then, by equation (1), we have

$$\Delta(0, j) = \Delta(0, n-1) + \Delta(n-1, j, l, j) + \Delta(n-1, j, 0, l) + \Delta(0, n-1, n-1, j).$$

Since the permutation is a-Monge, all numbers in the right-hand side of the above equality are non-negative. Moreover, we have $\Delta(n-1, j, l, j) = \Delta(j, n-1, j, l) > 0$. This implies that $\Delta(0, j) > \Delta(0, n-1)$, which is a contradiction with the choice of 0 and $n-1$.

Similarly, if $\langle l, 0, n-1, j\rangle$ is an a-Monge permutation, then there is a representation of $\Delta(l, j)$ as a sum of non-negative numbers which again include $\Delta(0, n-1)$ and $\Delta(n-1, j, l, j) > 0$. If $\langle l, n-1, 0, j\rangle$ is an a-Monge permutation, then there is a representation of $\Delta(l, j)$ which contains $\Delta(0, n-1)$ and $\Delta(0, j, l, n-1) > 0$.

Case 2 For any triple i, j, l, $i < j < l$, with $\Delta(i, j, j, l) < 0$, neither $i = 0$ nor $l = n-1$.

It is easy to see that this condition implies the following inequalities:

$\Delta(0, i, j, l) > 0$ because, otherwise, $\Delta(0, j, j, l) = \Delta(0, i, j, l) + \Delta(i, j, j, l) < 0$;

$\Delta(i, j, l, n-1) > 0$ because, otherwise, $\Delta(i, j, j, n-1) = \Delta(i, j, j, l) + \Delta(i, j, l, n-1) < 0$;

$\Delta(j, l, l, n-1) \geq 0$;

$\Delta(0, i, i, j) \geq 0$.

We claim that, given the above inequalities, at least one of $M[\{0, i, j, l\}]$ and $M[\{i, j, l, n-1\}]$ is a bad matrix.

We show first that if $\Delta(j, l) = 0$, then $M[\{0, i, j, l\}]$ is the bad matrix. Indeed, when trying to find an a-Monge permutation for this matrix, the assumption $0 \prec i$ yields $j \prec l$ (since $\Delta(0, i, j, l) > 0$) and $i \prec l$ (since $\Delta(0, i, i, l) = \Delta(0, i, i, j) + \Delta(0, i, j, l) > 0$). The constraint $j \prec l$ for the columns yields the constraint $j \prec i$, and, since $\Delta(i, l, j, l) = \Delta(i, j, j, l) + \Delta(j, l) = \Delta(i, j, j, l) < 0$, it also yields $l \prec i$. The contradictory precedence constraints $\{l \prec i, i \prec l\}$ prove that $M[\{0, i, j, l\}]$ is a bad matrix. So, we assume now that $\Delta(j, l) > 0$ (the case $\Delta(j, l) < 0$ is already eliminated).

By using a similar argument for the matrix $M[\{i, j, l, n-1\}]$, we see that that if $\Delta(i, j) = 0$ then this matrix is bad. So we will also assume that $\Delta(i, j) > 0$.

We now consider the submatrix $M[\{0, i, j, l\}]$ and will try to permute it into an a-Monge matrix. The assumption $0 \prec i$ yields $j \prec l$, $i \prec l$, $j \prec i$. Since $\Delta(i, j) > 0$, and so $\Delta(0, j, i, j) = \Delta(0, i, i, j) + \Delta(i, j, i, j) > 0$, we also have $j \prec 0$. This shows that a permutation other than $\langle j, 0, i, l\rangle$ cannot be an a-Monge permutation for $M[\{0, i, j, l\}]$. By analyzing the matrix $M[\{i, j, l, n-1\}]$ in a similar way, we see that the only potential a-Monge permutation for it is the permutation $\langle i, l, n-1, j\rangle$.

If $\langle j, 0, i, l\rangle$ is an a-Monge permutation for $M[\{0, i, j, l\}]$ then we must have $\Delta(0, i, j, 0) \geq 0$. Since $\Delta(0, i, i, j) \geq 0$ by the assumption of Case 2, and also $\Delta(0, i) \geq 0$, we have $\Delta(0, i, 0, j) = \Delta(0, i, 0, i) + \Delta(0, i, i, j) \geq 0$. However, $\Delta(0, i, 0, j) = -\Delta(0, i, j, 0)$, which implies that $\Delta(0, i) = 0$ and $\Delta(0, i, i, j) = 0$.

Moreover, if $\langle j, 0, i, l\rangle$ is an a-Monge permutation for $M[\{0, i, j, l\}]$ then $\Delta(j, i, i, l) \geq 0$. Similarly, if $\langle i, l, n-1, j\rangle$ is an a-Monge permutation for $M[\{i, j, l, n-1\}]$ then $\Delta(i, j, i, l) \geq 0$. But $\Delta(j, i, i, l) = -\Delta(i, j, i, l)$, so both are equal to 0.

If $\langle j,0,i,l\rangle$ is an a-Monge permutation for $M[\{0,i,j,l\}]$ then we must have $\Delta(j,0,i,l) \geq 0$. We can express $\Delta(j,0,i,l)$ as $\Delta(j,0,i,l) = -\Delta(0,j,i,l) = -(\Delta(0,i,i,j) + \Delta(0,i,j,l) + \Delta(i,j) + \Delta(i,j,j,l))$. Since $\Delta(0,i,i,j) = 0$ and $\Delta(i,j) + \Delta(i,j,j,l) = \Delta(i,j,i,l) = 0$, we get $\Delta(0,i,j,l) = -\Delta(j,0,i,l) \leq 0$. However, the inequality $\Delta(0,i,j,l) > 0$ is one of the four inequalities (see above) directly implied by the assumption of Case 2. Hence, we get a contradiction which proves that at least one of the matrices $M[\{0,i,j,l\}]$ and $M[\{i,j,l,n-1\}]$ is a bad matrix.

This completes the proof of the theorem. □

Note that the bound $|B| \leq 4$ in the above theorem is tight. Indeed, it can be straightforwardly checked that the following matrix is not permuted a-Monge, while any matrix obtained from it by deleting a row and a column (with the same index) is permuted a-Monge.

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

We can now generalize Theorem 5.3 to the case of several matrices.

Corollary 5.4 *Let $M_1,\ldots,M_m$ be $n \times n$ matrices. If there exists no permutation that simultaneously permutes all these matrices into a-Monge matrices, then there exists a subset of indices B with $|B| \leq 4$, such that no permutation of the indices in B simultaneously permutes matrices $M_1[B],\ldots,M_m[B]$ into a-Monge matrices.*

Proof: Consider the matrix $M = \sum_{i=1}^{m} M_i$. If M is not a permuted a-Monge matrix then, by Theorem 5.3, there exists a subset of indices B with $|B| \leq 4$, such that $M[B]$ is a bad matrix. Consider matrices $M_1[B],\ldots,M_m[B]$. If there existed a permutation that permutes all these matrices into a-Monge matrices, then the sum of the permuted matrices, which is $M[B]$, would be a permuted a-Monge matrix as well. This contradiction proves that there exists no permutation that simultaneously permutes matrices $M_1[B],\ldots,M_m[B]$ into a-Monge matrices.

Assume now that M is a permuted a-Monge matrix. The corresponding a-Monge permutation does not permute all of $M_1,\ldots,M_m$ into a-Monge matrices. Hence, there exist indices i,j,k,l and a pair of matrices, say, M_1 and M_2 such that $M_1(i,k) + M_1(j,l) - M_1(i,l) - M_1(j,k) > 0$ and $M_2(i,k) + M_2(j,l) - M_2(i,l) - M_2(j,k) < 0$. This implies that the matrices $M_1[\{i,j,k,l\}]$ and $M_2[\{i,j,k,l\}]$ cannot be simultaneously permuted into a-Monge matrices – this follows from the fact that the assumption $i \prec j$ implies $k \prec l$ for M_1 and $l \prec k$ for M_2. □

5.2 Reducing the number of matrices

We will now prove the bound $|\mathcal{F}'| \leq 3$ in Theorem 5.1, again via a-Monge matrices. In the proof, we will use special partial orders which we call *multipartite* partial orders.

We say that a partial order $\preceq$ on a set D is multipartite if and only if there is a partition of $D = D_1 \cup \ldots \cup D_t$, $t \geq 2$, such that $d \preceq d'$ if and only if $d = d'$ or else $d \in D_i$ and $d' \in D_j$ for some $1 \leq i < j \leq t$. If P is a multipartite order, then we will call the classes $D_1,\ldots,D_t$ the corresponding *partition classes* of P.

It is clear that if π is an a-Monge permutation for a matrix M then the reverse permutation π^- is also an a-Monge permutation for M. It is also clear that if M is an a-Monge matrix with at

least three rows, and the matrix obtained from M by simultaneously swapping rows s and t and columns s and t is again a-Monge then rows s and t are equivalent, i.e., $M(s,i) = M(t,i) + \alpha_{st}$, and columns s and t are equivalent as well. Note that swapping of equivalent rows and columns does not affect the property of being a-Monge. A matrix M is called Monge if $-M$ is a-Monge. It is shown in Observation 3.6 of [37] that if M is Monge, $i \prec j \prec k$ in M, and rows (columns) i, k are equivalent in M then row j is equivalent to these rows (columns). Clearly, the statement is also true for a-Monge matrices. Theorem 3.9 of [37] states that if a Monge matrix has no equivalent rows or columns then the only way to permute it to a Monge matrix is by using either the identity permutation id or its reverse id^-.

This leads to the following characterization of a-Monge permutations in terms of multipartite orders. For every anti-Monge square matrix M, there exist two mutually reverse multipartite orders such that a permutation (i.e. ordering) of the indices of M is an a-Monge permutation if and only if this ordering is an extension of one of the two multipartite orders. Two indices i, j belong to the same partition class of such a multipartite order if and only if both rows i, j and columns i, j are equivalent in M.

We will now prove two auxiliary lemmas about multipartite orders.

Lemma 5.5 *For any two multipartite orders P' and P'' on D, there are $a, b \in D$ such that a and b are comparable (not necessarily in the same direction) both in P' and in P''.*

Proof: Take a maximal chain in P'. If it is not entirely contained in a class of P'' then there are two elements in this chain belonging to two different classes of P'', that is, these elements are comparable both in P'' and in P'. If all elements in the maximal chain are contained in the same class of P'', then pick any element d in a different class of P''. This element is comparable, in P'', with all elements from the chain, and, clearly, it is comparable with at least one of these elements in P'. □

Let us say that a collection $\mathcal{P} = \{P_1, \ldots, P_l\}$ of multipartite orders is *conflicting* if their union (considered as a digraph $G_{\mathcal{P}}$) contains a directed cycle.

Lemma 5.6 *If a collection $\mathcal{P} = \{P_1, \ldots, P_l\}$ is conflicting then the digraph $G_{\mathcal{P}}$ contains arcs (a, b) and (b, a) for some distinct a, b.*

Proof: Let $a_1, \ldots, a_t, a_1$ be a shortest directed cycle in $G_{\mathcal{P}}$, and assume, for contradiction, that $t > 2$. Without loss of generality, let $(a_1, a_2) \in P_1$. In this case, $(a_2, a_3) \notin P_1$, since, otherwise, we would have $(a_1, a_3) \in P_1$ and get a shorter cycle. Without loss of generality, assume that $(a_2, a_3) \in P_2$. Since the order P_1 is multipartite, we conclude that a_1 and a_3 are comparable in P_1. Furthermore, since we cannot have $(a_1, a_3) \in P_1$, we have $(a_3, a_1) \in P_1$. Since $(a_1, a_2) \in P_1$, the transitivity of P_1 implies that $(a_3, a_2) \in P_1$, which, together with $(a_2, a_3) \in P_2$, gives us the required arcs. □

Proposition 5.7 *Let $U = \{M_1, \ldots, M_m\}$ be a set of matrices of size $n \times n$ such that no permutation is an a-Monge permutation for all matrices in U. Then, there is a subset $U' \subseteq U$ such that $|U'| \leq 3$ and no permutation is an a-Monge permutation for all matrices in U'.*

Proof: We may assume that every matrix in U is a permuted a-Monge matrix, since, otherwise, the result follows immediately. Start with matrix $M_1 \in U$ and choose any of the two multipartite orders that describe the set of corresponding a-Monge permutations for M_1. Call this order P_1. By

Lemma 5.5, there is a pair $(a, b) \in P_1$ such that $a \neq b$ and a and b are comparable in P_2, where P_2 is the multipartite order for M_2. We may assume that $(a, b) \in P_2$, since, otherwise, the other multipartite order for P_2 would be chosen.

If there is a pair of distinct elements (c, d) such that $(c, d) \in P_1$ and $(d, c) \in P_2$, then there exists no a-Monge permutation for M_1 and M_2 and the proposition is proved. So we may assume that $\{P_1, P_2\}$ is not conflicting. Since P_1 shares a pair of comparable elements with any multipartite order, we can in the same way choose a multipartite order P_i for each matrix M_i. If, for some i, the pair $\{P_1, P_i\}$ is conflicting, then the proposition is proved. So assume that all such pairs of orders are non-conflicting. Note that if we chose the other multipartite order for M_1, this would have led to choosing the other multipartite orders for all $M_1, \ldots, M_m$.

Since there is no common a-Monge permutation for all of $M_1, \ldots, M_m$, we know that the collection $\{P_1, \ldots, P_m\}$ of orders that we have constructed is conflicting. By Lemma 5.6, there are orders P_i and P_j such that, for some distinct e, f, we have $(e, f) \in P_i$ and $(f, e) \in P_j$. Since both P_i and P_j share with P_1 some pairs of elements comparable in the same direction, we conclude that there is no common a-Monge permutation for M_1, M_i, M_j. This completes the proof. □

Note that the bound $|U'| \leq 3$ in the above proposition is tight. Indeed, each of the following three matrices is permuted a-Monge, every two of them have a common a-Monge permutation, but there is no common a-Monge permutation for all three of them.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

6 Main result

We will need the following two technical lemmas. They will be used in our hardness proof to reduce the argument to the case when all non-unary predicates are binary and their matrices do not contain all-ones rows or columns.

Lemma 6.1 *If $\mathcal{F}$ is not supermodular on any chain on D then $\mathcal{F} \cup \mathcal{U}_D$ can strictly implement a collection $\mathcal{F}'$ of binary predicates which is is not supermodular on any chain on D.*

Proof: Let $f \in \mathcal{F}$ be *not* supermodular on some fixed chain. By Observation 4.3(2), f is n-ary with $n \geq 2$. By Lemma 4.5, it is possible to substitute constants for some $n-2$ variables of f to obtain a binary predicate f' which is not supermodular on this chain. Assume without loss of generality that these variables are the last $n-2$ variables, and the corresponding constants are $d_3, \ldots, d_n$, that is, $f'(x, y) = f(x, y, d_3, \ldots, d_n)$. Then the following is a strict $(n-1)$-implementation of f':

$$f'(x, y) + (n-2) = \max_{z_3, \ldots, z_n} [f(x, y, z_3, \ldots, z_n) + u_{\{d_3\}}(z_3) + \ldots + u_{\{d_n\}}(z_n)].$$

Repeating this for all chains on D, one can strictly implement a collection $\mathcal{F}'$ of binary predicates that is not supermodular on any chain. □

Lemma 6.2 *[Lemma 3.3 [28]] Assume that $h \in R_D^{(2)}$ and there is $a \in D$ such that $h(x, a) = 1$ for all $x \in D$. Let $h'(x, y) = 0$ if $y = a$ and $h'(x, y) = h(x, y)$ if $y \neq a$. Then the following holds:*

1. *for any chain on D, h and h' are supermodular (or not supermodular) on the chain simultaneously;*

2. the problems Max CSP($\{h\} \cup \mathcal{U}_D$) *and* Max CSP($\{h'\} \cup \mathcal{U}_D$) *are AP-reducible to each other.*

Recall that all predicates from $\mathcal{C}_D$ are supermodular on any chain on D. Moreover, it is shown in (the proof of) Lemma 5.1 of [11] that all predicates from $\mathcal{C}_D$ are supermodular on a lattice if and only if the lattice is a chain.

We will now prove our main result:

Theorem 6.3 *If $\mathcal{F}$ is supermodular on some chain on D then weighted* Max CSP($\mathcal{F} \cup \mathcal{C}_D$) *belongs to* **PO**. *Otherwise,* Max CSP($\mathcal{F} \cup \mathcal{C}_D$) $- B$ *is* **APX**-*complete.*

Proof: The tractability part of the proof follows immediately from Theorem 4.10 (see also Observation 4.3(1)). By Lemmas 3.2 and 3.4, it is sufficient to prove the hardness part for sets of the form $\mathcal{F} \cup \mathcal{U}_D$. We will show that $\{neq_2\}$ can be obtained from $\mathcal{F} \cup \mathcal{U}_D$ by using the following two operations:

1. replacing $\mathcal{F} \cup \mathcal{U}_D$ by a subset of $\mathcal{F} \cup \mathcal{U}_D \cup \{f\}$ where f is a predicate that can be strictly implemented from $\mathcal{F} \cup \mathcal{U}_D$;
2. replacing $\mathcal{F} \cup \mathcal{U}_D$ by a subset of $\mathcal{F}|_{D'} \cup \mathcal{U}_{D'}$ for some D'.

By Example 2.5 and Lemmas 3.4 and 3.5, this will establish the result.

It follows from Lemmas 6.1 and 3.4 that it is sufficient to prove the hardness part of Theorem 6.3 assuming that $\mathcal{F}$ contains only binary predicates. Now, Theorem 5.1 and Lemma 3.5 imply that, in addition, we can assume that $|\mathcal{F}| \leq 3$ and $|D| \leq 4$. Note that the case $|D| \leq 3$ is already considered in Corollary 4.9 (see also Remark 4.8), so it remains to consider the case $|D| = 4$; we can without loss of generality assume in the rest of the proof that $D = \{0, 1, 2, 3\}$. Moreover, due to Lemma 3.5, we may consider only sets $\mathcal{F}$ satisfying the following condition:

$$\text{for any proper subset } D' \subset D,\ \mathcal{F}|_{D'} \text{ is supermodular on some chain on } D'. \qquad (*)$$

We can assume that $\mathcal{F}$ is minimal with respect to inclusion, that is, every proper non-empty subset of $\mathcal{F}$ is supermodular on some chain on D. We will consider three cases depending on the number of predicates in $\mathcal{F}$. Note that, by Lemma 6.2, we can without loss of generality assume that none of the predicates in $\mathcal{F}$ has a matrix containing an all-ones row or column (this property does not depend on the order of indices in the matrix).

We prove the result by using a simple computer-generated enumeration in each of the three cases. In each case, we first produce a list of all possible sets $\mathcal{F}$ with the above restrictions, then reduce the list by using some obvious symmetries (such as isomorphism and anti-isomorphism), and, finally, for each remaining set $\mathcal{F}$, provide a strict implementation of a set $\mathcal{F}'$ that is known to have an **APX**-hard Max CSP($\mathcal{F}'$) problem. To compactly describe such symmetries, we introduce some notation. Let π be a permutation on D and f a binary predicate on D. Then, we define $\pi(f)$ to be the predicate such that $\pi(f)(a, b) = 1$ if and only if $f(\pi(a), \pi(b)) = 1$ for all $a, b \in D$; we say that the predicate $\pi(f)$ is isomorphic to f. We also define the predicate f^t so that $f^t(a, b) = 1$ if and only if $f(b, a) = 1$ for all $a, b \in D$ (this corresponds to transposing the matrix of f). We say that a predicate of the form $\pi(f^t)$ is anti-isomorphic to f.

Case 1. $|\mathcal{F}| = 1$.

First, we use exhaustive search to generate the list of all binary predicates f on D that (a) do not have all-ones rows or columns, (b) are not supermodular on any chain on D, and (c) $\mathcal{F} = \{f\}$

satisfies condition $(*)$. Moreover, we may consider predicates only up to isomorphism and anti-isomorphism. Thus, this list is then processed as follows: for every predicate f in the list, in order, remove all predicates below f in the list that are isomorphic or anti-isomorphic to f.

Clearly, it is sufficient to prove the hardness result for all predicates that remain in the optimized list. Since there are only $2^{16} = 65536$ predicates to check, it is clear that generating and optimizing the list can easily be (and actually was) performed by a computer. The optimized list contains only 27 predicates which are given in Fig. 3.

$$h'_1 \begin{matrix}1000\\0110\\1000\\0000\end{matrix} \quad h'_2 \begin{matrix}1000\\1101\\1000\\0000\end{matrix} \quad h'_3 \begin{matrix}1001\\0111\\1110\\1001\end{matrix} \quad h'_4 \begin{matrix}1010\\0101\\1010\\1000\end{matrix} \quad h'_5 \begin{matrix}1010\\0110\\0000\\0000\end{matrix} \quad h'_6 \begin{matrix}1010\\0111\\1010\\1000\end{matrix} \quad h'_7 \begin{matrix}1010\\0111\\1110\\1000\end{matrix} \quad h'_8 \begin{matrix}1011\\0101\\1010\\0000\end{matrix} \quad h'_9 \begin{matrix}1011\\0111\\0010\\0000\end{matrix} \quad h'_{10} \begin{matrix}1011\\0111\\0010\\0001\end{matrix} \quad h'_{11} \begin{matrix}1011\\0111\\0011\\0000\end{matrix} \quad h'_{12} \begin{matrix}1011\\0111\\0110\\1001\end{matrix} \quad h'_{13} \begin{matrix}1011\\0111\\1010\\0000\end{matrix}$$

$$h'_{14} \begin{matrix}1011\\0111\\1010\\0001\end{matrix} \quad h'_{15} \begin{matrix}1011\\0111\\1110\\0000\end{matrix} \quad h'_{16} \begin{matrix}1011\\0111\\1110\\0001\end{matrix} \quad h'_{17} \begin{matrix}1011\\0111\\1110\\1001\end{matrix} \quad h'_{18} \begin{matrix}1011\\0111\\1110\\1101\end{matrix} \quad h'_{19} \begin{matrix}1011\\1101\\1010\\0000\end{matrix} \quad h'_{20} \begin{matrix}1100\\1101\\1000\\0000\end{matrix} \quad h'_{21} \begin{matrix}1101\\0110\\0110\\1001\end{matrix} \quad h'_{22} \begin{matrix}1101\\1100\\0010\\0000\end{matrix} \quad h'_{23} \begin{matrix}1101\\1110\\0000\\0000\end{matrix} \quad h'_{24} \begin{matrix}1101\\1110\\0110\\1001\end{matrix} \quad h'_{25} \begin{matrix}1110\\1100\\0000\\0000\end{matrix} \quad h'_{26} \begin{matrix}1110\\1100\\1010\\0000\end{matrix}$$

$$h'_{27} \begin{matrix}1110\\1101\\1010\\0000\end{matrix}$$

Figure 3: The optimized list of 27 predicates from the proof of Case 1. The predicates are represented by tables of values.

We show, starting from h'_1 and proceeding in order, that $\{h'_i\} \cup \mathcal{U}_D$ strictly implements some binary predicate g such that either, for some $D' \subset D$, the predicate $g|_{D'}$ is not supermodular on any chain on D' or g is equal to h'_j for some $j < i$ (up to isomorphism and anti-isomorphism). These implementations can be found in Appendix A. This, together with Remark 4.8, implies that neq_2 can be obtained from $\mathcal{F} \cup \mathcal{U}_D$.

Case 2. $|\mathcal{F}| = 2$.

Let $\mathcal{F} = \{f_1, f_2\}$. As in Case 1, we use exhaustive search to generate the list of all pairs of binary predicates on D such that (a) they do not have all-ones rows or columns, (b) each of the two predicates is supermodular on at least one chain, but there is no chain on which they are both supermodular, and (c) $\mathcal{F}$ satisfies condition $(*)$. Without loss of generality, we can assume that f_1 is supermodular on the chain $0 < 1 < 2 < 3$, that is, the matrix of f_1 with this order of indices is a-Monge. Since the matrix of f_1 does not have all-ones row or column, its structure is described in Lemma 4.4. Similarly, the matrix of f_2 is a permuted a-Monge matrix, since $\pi(f_2)$ is supermodular for some permutation π.

We can also assume that the a-Monge matrices for f_1 and f_2 (with respect to the orders on which the predicates are supermodular) have the third form $(L_4^{pq} + R_4^{st})$ from Lemma 4.4. The reason is that if, say, the matrix of f_1 has the form L_4^{pq} for some $0 \le p, q \le 2$ then $f'_1(x, y) + 1 = f_1(x, y) + u_{\{p+1,\dots,3\}}(x) + u_{\{q+1,\dots,3\}}(y)$ is a strict 2-implementation of the predicate f' whose matrix is $R_4^{(p+1)(q+1)}$. Moreover, $f''_1(x, y) = f_1(x, y) + f'_1(x, y)$ is a strict 1-implementation of a predicate whose matrix is $L_4^{pq} + R_4^{(p+1)(q+1)}$. Hence, we can replace f_1 by f''_1 in this pair, and show the hardness result for $\{f''_1, f_2\}$.

It is clear that if we prove the result for *all* pairs (f_1, f_2) with some fixed f_1, then this also proves the result for all pairs with the first component f_1^t, or $\pi(f_1)$, or $\pi(f_1^t)$ where $\pi(x) = 3 - x$. This implies that it is sufficient to consider only predicates from Fig. 1 as possible candidates for f_1. Moreover, it can be straightforwardly checked by using a computer that if f_1 is one of the predicates $h_1, h_3, h_4, h_6, h_7, h_9, h_{10}$ from Fig. 1, then $\mathcal{F} = \{f_1, f_2\}$ fails to satisfy condition $(*)$.

Hence, all pairs (f_1, f_2), where at least one of f_1 and $\pi(f_2)$ (for some permutation π) coincides with one of 7 predicates above, will not be on the list of pairs that we need to consider.

Obviously, if we prove the result for some pair (f_1, f_2) then this also proves the result for (f_1, f_2^t). Hence, provided $f_2 \neq f_2^t$, one of these two pairs can be excluded from the list.

Now we show that predicates h_5, h_{11}, h_{12}, and h_{17} from Fig. 1 can also be excluded from consideration because they can strictly implement some other predicates from Fig. 1. Implementations:

$$\left\{ f := \begin{smallmatrix} 1100 \\ 1101 \\ 1101 \\ 0001 \end{smallmatrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_6 \begin{smallmatrix} 1100 \\ 0001 \\ 0001 \\ 0001 \end{smallmatrix} =: g \qquad f = h_{17}, g = \pi(h_8) \text{ where } \pi(x) = 3 - x$$
$$g(x,y) + 5 = max_{z,w}[f(z,w) + f(z,y) + f(x,z) + f(x,w) + u_{\{0,3\}}(z) + u_{\{3\}}(w) + u_{\{0\}}(x)]$$

$$\left\{ f := \begin{smallmatrix} 1110 \\ 1110 \\ 0001 \\ 0001 \end{smallmatrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_3 \begin{smallmatrix} 1110 \\ 1110 \\ 0000 \\ 0001 \end{smallmatrix} =: g \qquad f = h_{11}, g = h_5$$
$$g(x,y) + 2 = max_z[f(z,x) + f(z,y) + f(x,z)]$$

$$\left\{ f := \begin{smallmatrix} 1110 \\ 1110 \\ 0000 \\ 0001 \end{smallmatrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_2 \begin{smallmatrix} 1100 \\ 1100 \\ 1101 \\ 0001 \end{smallmatrix} =: g \qquad f = h_5, g = h_{16}$$
$$g(x,y) + 1 = max_z[f(x,z) + f(y,z) + u_{\{2\}}(x)]$$

$$\left\{ f := \begin{smallmatrix} 1110 \\ 0001 \\ 0001 \\ 0001 \end{smallmatrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_4 \begin{smallmatrix} 1000 \\ 1001 \\ 1001 \\ 0001 \end{smallmatrix} =: g \qquad f = h_{12}, g = h_{15}$$
$$g(x,y) + 3 = max_z[f(z,x) + f(z,y) + f(x,z) + f(y,z) + u_{\{1\}}(z) + u_{\{1,2\}}(x)]$$

As above, all pairs (f_1, f_2) such that, for some permutation π, $\pi(f_2)$ or $\pi(f_2^t)$ is one of $h_5, h_{11}, h_{12}, h_{17}$, can also be excluded from the list.

Finally, we can exclude from the list all pairs isomorphic to some pair higher up in the list. That is, we exclude pair (f_1, f_2) if there is a permutation π such that either $\pi(f_1) = f_1$ and the pair $(f_1, \pi(f_2))$ is above (f_1, f_2) in the list or if there is a permutation π such that the pair $(\pi(f_2), \pi(f_1))$ is above (f_1, f_2) in the list (in the latter case, $\pi(f_2)$ must be supermodular on $0 < 1 < 2 < 3$).

The optimized list now contains 27 pairs of predicates. In Appendix B, we provide strict implementations for them that show that, for each pair (f_1, f_2) in this list, $\{f_1, f_2\} \cup \mathcal{U}_D$ implements either a pair above it in the list or else a binary predicate g such that, for some $D' \subset D$, the predicate $g|_{D'}$ is not supermodular on any chain on D'. As in Case 1, it follows that neq_2 can be obtained from $\mathcal{F} \cup \mathcal{U}_D$.

Case 3. $|\mathcal{F}| = 3$.

It can be checked by computer-assisted exhaustive search that there does not exist such a set $\mathcal{F}$. Simply loop through all triples of (not necessarily distinct) binary predicates on $\{0, 1, 2\}$ which are supermodular on the chain $0 < 1 < 2$ and check that each possible extension to a triple of pairwise distinct predicates on D results in a set $\mathcal{F}$ satisfying one of the following conditions:

1. $\mathcal{F}$ is supermodular on some chain on D,
2. for some $D' \subset D$, $\mathcal{F}|_{D'}$ is not supermodular on any chain on D',
3. some proper subset of $\mathcal{F}$ is not supermodular on any chain on D.

□

Remark 6.4 *There are two main ways to represent a predicate: by its complete table of values and by the set of tuples which satisfy the predicate. By using Lemma 4.5 and Corollary 5.4, it is not hard to show that if the former representation is used or if D is fixed, then it can be checked in polynomial time whether a given (finite) $\mathcal{F}$ is supermodular on some chain on D. However, if D is not fixed and the latter representation is used then it is an open question whether there exists a polynomial-time algorithm for checking supermodularity of $\mathcal{F}$ on some chain on D.*

7 Application to LIST H-COLOURING optimization

Recall that a *homomorphism* from a digraph $G = (V_G, A_G)$ to a digraph $H = (V_H, A_H)$ is a mapping $\varphi : V_G \to V_H$ such that $(\varphi(v), \varphi(w)) \in A_H$ whenever $(v, w) \in A_G$. In this case, the digraph G is said to be *H-colourable*. The GRAPH H-COLOURABILITY problem is, given a digraph G, to decide whether it is H-colourable. This problem attracts much attention in graph theory [25].

In this section, we consider the case when $\mathcal{F}$ consists of a single binary predicate h. This predicate specifies a digraph H such that $V_H = D$ and (u, v) is an arc in H if and only if $h(u, v) = 1$. Any instance $\mathcal{I} = (V, C)$ of CSP($\{h\}$) can be associated with a digraph $G_{\mathcal{I}}$ whose nodes are the variables in V and whose arcs are the scopes of constraints in C. It is not difficult to see that the question whether all constraints in $\mathcal{I}$ are simultaneously satisfiable is equivalent to the question whether $G_{\mathcal{I}}$ is H-colourable. Therefore, the problem CSP($\{h\}$) is precisely the GRAPH H-COLOURABILITY problem for the digraph H. The problems CSP($\{h\} \cup \mathcal{U}_D$) and CSP($\{h\} \cup \mathcal{C}_D$) are equivalent to the LIST H-COLOURING and H-RETRACTION problems, respectively. In the former problem, every vertex of an input digraph G gets a list of allowed target vertices in H, and the question is whether G has an H-colouring subject to the list constraints. The latter problem is the same except that each list contains either one vertex or all vertices of H. These problems also attract much attention in graph theory [25].

The problem MAX CSP($\{h\} \cup \mathcal{U}_D$) can then be viewed as the LIST H-COLOURING optimization problem: for every vertex v of an input digraph G, there is a list $L_v \subseteq V_H$ along with a function $\rho_v : L_v \to \mathbb{Z}^+$ that indicates the 'score' which a mapping $V_G \to V_H$ gets if it sends v to a certain vertex (if a mapping sends v to a vertex outside of L_v then this adds nothing to the 'cost' of this mapping). Then the goal is to maximize the combined 'cost' of such a mapping which is obtained by adding weights of preserved arcs and 'scores' from the lists. The 'score' functions ρ_v arise as the result of the possible presence in C of several weighted constraints of the form $u_{D'}(v)$ for different $D' \subseteq D$ and the same v. Thus, Theorem 6.3 in the case when $\mathcal{F} = \{h\}$ presents a complexity classification of list H-colouring optimization problems. Digraphs H corresponding to the tractable cases of this problem are the digraphs that have an a-Monge adjacency matrix under some total ordering on V_H (note that this property of digraphs can be recognised in polynomial time, e.g., by using Theorem 5.3). Such matrices without all-one rows or columns are described in Lemma 4.4. It remains to note that, as is easy to see, replacing either some all-zero row or some all-zero columns with all-one ones does not affect the property of being a-Monge.

We remark that another problem related to optimizing list homomorphisms between graphs was recently considered in [19], in connection with some problems arising in defence logistics.

Acknowledgements

The authors are thankful to Gerhard Woeginger for encouraging this collaboration and to Johan Håstad for suggesting to use the bounded occurrence property in our proofs. The authors would also like to thank the anonymous referees for providing useful comments on the paper.

References

[1] P. Alimonti and V. Kann. Some APX-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1-2):123–134, 2000.

[2] G. Ausiello, P. Creszenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer, 1999.

[3] C. Bazgan and M. Karpinski. On the complexity of global constraint satisfaction. In *ISAAC'05*, pages 624–633, 2005.

[4] P. Berman and M. Karpinski. Improved approximation lower bounds on small occurrence optimization. Technical Report TR03-008, Electronic Colloquium on Computational Complexity (ECCC), 2003.

[5] F. Börner, A. Bulatov, P. Jeavons, and A. Krokhin. Quantified constraints: Algorithms and complexity. In *CSL'03*, volume 2803 of *LNCS*, pages 58–70, 2003.

[6] A. Bulatov. Tractable conservative constraint satisfaction problems. In *LICS'03*, pages 321–330, 2003.

[7] A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006.

[8] A. Bulatov and V. Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. *Information and Computation*, 205(5):651–678, 2007.

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

[10] R.E. Burkard, B. Klinz, and R. Rudolf. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics*, 70:95–161, 1996.

[11] 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.

[12] N. Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *Journal of Computer and System Sciences*, 51:511–522, 1995.

[13] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*, volume 7 of *SIAM Monographs on Discrete Mathematics and Applications*. 2001.

[14] M. Datar, T. Feder, A. Gionis, R. Motwani, and R. Panigrahy. A combinatorial algorithm for MAX CSP. *Information Processing Letters*, 85(6):307–315, 2003.

[15] V.G. Deineko, R. Rudolf, and G.J. Woeginger. A general approach to avoiding 2×2 submatrices. *Computing*, 52:371–388, 1994.

[16] L. Engebretsen. The non-approximability of non-Boolean predicates. *SIAM Journal on Discrete Mathematics*, 18(1):114–129, 2004.

[17] T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1998.

[18] S. Fujishige. *Submodular Functions and Optimization*, volume 58 of *Annals of Discrete Mathematics*. Elsevier, 2nd edition, 2005.

[19] 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.

[20] G. Hast. *Beating a random assignment: Approximating constraint satisfaction problems*. PhD thesis, Royal Institute of Technology, Stockholm, 2005.

[21] J. Håstad. On bounded occurrence constraint satisfaction. *Information Processing Letters*, 74(1-2):1–6, 2000.

[22] J. Håstad. Some optimal inapproximability results. *J. ACM*, 48:798–859, 2001.

[23] J. Håstad. Every 2-CSP allows nontrivial approximation. In *Proceedings of STOC'05*, pages 740–746, 2005.

[24] P. Hell. Algorithmic aspects of graph homomorphisms. In C. Wensley, editor, *Surveys in Combinatorics 2003*, volume 307 of *LMS Lecture Note Series*, pages 239 – 276. Cambridge University Press, 2003.

[25] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.

[26] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, 2001.

[27] P. Jonsson. Boolean constraint satisfaction: Complexity results for optimization problems with arbitrary weights. *Theoretical Computer Science*, 244(1-2):189–203, 2000.

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

[29] 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.

[30] M. Karpinski. Approximating bounded degree instances of NP-hard problems. In *Proceedings 13th Conference on Fundamentals of Computation Theory, FCT'01*, volume 2138 of *Lecture Notes in Computer Science*, pages 24–34. Springer-Verlag, 2001.

[31] S. Khanna, M. Sudan, L. Trevisan, and D. Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30(6):1863–1920, 2001.

[32] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for Max-Cut and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007.

[33] B. Klinz, R. Rudolf, and G. Woeginger. Permuting matrices to avoid forbidden submatrices. *Discrete Applied Mathematics*, 60:223–248, 1995.

[34] A. Krokhin, A. Bulatov, and P. Jeavons. The complexity of constraint satisfaction: an algebraic approach. In *Structural Theory of Automata, Semigroups, and Universal Algebra*, volume 207 of *NATO Science Series II: Math., Phys., Chem.*, pages 181–213. Springer Verlag, 2005.

[35] A. Krokhin and B. Larose. Maximum constraint satisfaction on diamonds. In *CP'05*, volume 3709 of *LNCS*, pages 388–402, 2005.

[36] I. Pe'er, T. Pupko, R. Shamir, and R. Sharan. Incomplete directed perfect phylogeny. *SIAM Journal on Computing*, 33(3):590–607, 2004.

[37] R. Rudolf. Recognition of d-dimensional Monge arrays. *Discrete Applied Mathematics*, 52(1):71–82, 1994.

[38] T.J. Schaefer. The complexity of satisfiability problems. In *STOC'78*, pages 216–226, 1978.

[39] A. Schrijver. A combinatorial algorithm minimizing submodular functions in polynomial time. *Journal of Combinatorial Theory, Ser.B*, 80:346–355, 2000.

[40] D. Topkis. *Supermodularity and Complementarity*. Princeton University Press, 1998.

[41] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

Appendix A: Strict implementations from Case 1

It is assumed throughout that $D = \{0, 1, 2, 3\}$. Implementations should be read as follows:

- the symbol $\stackrel{s}{\Longrightarrow}_\alpha$ means "strictly α-implements";
- $\mathcal{U}$ always denotes $\mathcal{U}_D$;
- $Y = \{x, y\}$ is the set of primary variables and $Z = \{z, w\}$ is the set of auxiliary variables (see Definition 3.1).

Each implementation produces some predicate g such that either g or $\pi(g)$, or $\pi(g^c)$ (for some permutation π) is a predicate for which a strict implementation has already been found, or else a predicate g such that, for some $D' \subset D$, $g|_{D'}$ is not supermodular on any chain on D'. We will describe the latter situation by writing, for simplicity, that "$g|_{D'}$ is bad". If $|D'| = 2$ then one can directly verify that the corresponding matrix is not a-Monge (there is no need to permute rows and columns). For the case $|D'| = 3$, one can use Lemma 4.4 to quickly check that the matrix of $g|_{D'}$ is not a permuted a-Monge matrix.

1. $\left\{ h'_1 := \begin{matrix} 1000 \\ 0110 \\ 1000 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_2 \begin{matrix} 1000 \\ 1110 \\ 1000 \\ 0000 \end{matrix} =: g \qquad g|_{\{0,1,3\}}$ is bad

$g(x, y) + 1 = max_z[h'_1(z, y) + h'_1(x, z) + u_{\{3\}}(z)]$

2. $\left\{ h'_2 := \begin{matrix} 1000 \\ 1101 \\ 1000 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_3 \begin{matrix} 1000 \\ 1101 \\ 1010 \\ 0000 \end{matrix} =: g \qquad g|_{\{0,2,3\}}$ is bad

$g(x, y) + 2 = max_z[h'_2(z, x) + h'_2(z, y) + h'_2(x, y) + u_{\{3\}}(z) + u_{\{2\}}(x) + u_{\{2\}}(y)]$

3. $\left\{ h'_3 := \begin{matrix} 1001 \\ 0111 \\ 1110 \\ 1001 \end{matrix} \right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_4 \begin{matrix} 1000 \\ 0111 \\ 1110 \\ 0001 \end{matrix} =: g \qquad g|_{\{0,1,3\}}$ is bad

$g(x, y) + 3 = max_z[h'_3(z, x) + h'_3(z, y) + h'_3(x, y) + u_{\{1,2\}}(z)]$

4. $\left\{ h'_4 := \begin{matrix} 1010 \\ 0101 \\ 1010 \\ 1000 \end{matrix} \right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_4 \begin{matrix} 1101 \\ 0101 \\ 1000 \\ 0101 \end{matrix} =: g \qquad g|_{\{0,1,2\}}$ is bad

$g(x, y) + 3 = max_{z,w}[h'_4(z, w) + h'_4(z, y) + h'_4(w, x) + u_{\{1,3\}}(z)]$

5. $\left\{ h'_5 := \begin{matrix} 1010 \\ 0110 \\ 0000 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_3 \begin{matrix} 1000 \\ 0100 \\ 0001 \\ 0001 \end{matrix} =: g \qquad g|_{\{0,1,3\}}$ is bad

$g(x, y) + 2 = max_z[h'_5(x, z) + h'_5(x, y) + h'_5(y, z) + u_{\{3\}}(z) + u_{\{2,3\}}(x) + u_{\{3\}}(y)]$

6. $\left\{ h'_6 := \begin{matrix} 1010 \\ 0111 \\ 1010 \\ 1000 \end{matrix} \right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_4 \begin{matrix} 1010 \\ 0111 \\ 1010 \\ 1001 \end{matrix} =: g \qquad g|_{\{0,1,3\}}$ is bad

$g(x, y) + 3 = max_z[h'_6(x, z) + h'_6(x, y) + h'_6(y, z) + u_{\{2\}}(z) + u_{\{3\}}(x) + u_{\{3\}}(y)]$

7. $\left\{ h'_7 := \begin{matrix} 1010 \\ 0111 \\ 1110 \\ 1000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix} 1110 \\ 0000 \\ 0000 \\ 1010 \end{matrix}$ $=: g$ $\quad g|_{\{2,3\}}$ is bad

$g(x,y) + 2 = max_z[h'_7(z,y) + h'_7(x,z) + u_{\{0,3\}}(x)]$

8. $\left\{ h'_8 := \begin{matrix} 1011 \\ 0101 \\ 1010 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_6 \begin{matrix} 1010 \\ 0000 \\ 1010 \\ 1011 \end{matrix}$ $=: g$ $\quad g|_{\{0,1,3\}}$ is bad

$g(x,y) + 5 = max_{z,w}[h'_8(z,w) + h'_8(z,x) + h'_8(z,y) + h'_8(w,x) + u_{\{2\}}(z) + u_{\{0\}}(w) + u_{\{1,3\}}(x)]$

9. $\left\{ h'_9 := \begin{matrix} 1011 \\ 0111 \\ 0010 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix} 1001 \\ 0101 \\ 0010 \\ 1101 \end{matrix}$ $=: g$ $\quad g|_{\{0,1,2\}}$ is bad

$g(x,y) + 2 = max_z[h'_9(z,x) + h'_9(x,y) + h'_9(y,z) + u_{\{3\}}(z) + u_{\{3\}}(x) + u_{\{3\}}(y)]$

10. $\left\{ h'_{10} := \begin{matrix} 1011 \\ 0111 \\ 0010 \\ 0001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix} 1011 \\ 0111 \\ 0010 \\ 0000 \end{matrix}$ $=: g$ $\quad g = h'_9$

$g(x,y) + 2 = max_z[h'_{10}(z,x) + h'_{10}(z,y) + u_{\{2\}}(z) + u_{\{0,1\}}(x)]$

11. $\left\{ h'_{11} := \begin{matrix} 1011 \\ 0111 \\ 0011 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_2 \begin{matrix} 1000 \\ 0100 \\ 0000 \\ 1100 \end{matrix}$ $=: g$ $\quad \pi(g^t) = h'_5$ where $\pi(0,1,2,3) = (0,1,3,2)$

$g(x,y) + 1 = h'_{11}(x,y) + u_{\{3\}}(x) + u_{\{0,1\}}(y)$

12. $\left\{ h'_{12} := \begin{matrix} 1011 \\ 0111 \\ 0110 \\ 1001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix} 0110 \\ 0111 \\ 0111 \\ 0000 \end{matrix}$ $=: g$ $\quad g|_{\{0,1,3\}}$ is bad

$g(x,y) + 2 = max_z[h'_{12}(z,y) + h'_{12}(x,z) + u_{\{1,2\}}(z)]$

13. $\left\{ h'_{13} := \begin{matrix} 1011 \\ 0111 \\ 1010 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix} 1011 \\ 0101 \\ 1010 \\ 0000 \end{matrix}$ $=: g$ $\quad g = h'_8$

$g(x,y) + 2 = max_z[h'_{13}(z,x) + h'_{13}(x,y) + h'_{13}(y,z) + u_{\{3\}}(z) + u_{\{3\}}(y)]$

14. $\left\{ h'_{14} := \begin{matrix} 1011 \\ 0111 \\ 1010 \\ 0001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix} 0001 \\ 0111 \\ 0000 \\ 0001 \end{matrix}$ $=: g$ $\quad \pi(g) = h'_2$ where $\pi(0,1,2,3) = (3,1,0,2)$

$g(x,y) + 2 = max_z[h'_{14}(z,y) + h'_{14}(x,z) + u_{\{1,3\}}(z)]$

15. $\left\{ h'_{15} := \begin{matrix} 1011 \\ 0111 \\ 1110 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_4 \begin{matrix} 1011 \\ 0101 \\ 1010 \\ 0000 \end{matrix}$ $=: g$ $\quad g = h'_8$

$g(x,y) + 3 = max_z[h'_{15}(x,z) + h'_{15}(x,y) + h'_{15}(y,z) + u_{\{0,3\}}(z) + u_{\{3\}}(x) + u_{\{3\}}(y)]$

16. $\left\{ h'_{16} := \begin{matrix} 1011 \\ 0111 \\ 1110 \\ 0001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_4 \begin{matrix} 1011 \\ 0000 \\ 1011 \\ 0001 \end{matrix} =: g \qquad g|_{\{0,1,3\}} \text{ is bad}$

$g(x,y) + 3 = max_z[h'_{16}(z,x) + h'_{16}(z,y) + h'_{16}(x,z) + u_{\{0,3\}}(z)]$

17. $\left\{ h'_{17} := \begin{matrix} 1011 \\ 0111 \\ 1110 \\ 1001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_4 \begin{matrix} 1010 \\ 0111 \\ 1110 \\ 0001 \end{matrix} =: g \qquad g|_{\{0,1,3\}} \text{ is bad}$

$g(x,y) + 3 = max_z[h'_{17}(z,x) + h'_{17}(z,y) + h'_{17}(x,y) + u_{\{1,2\}}(z)]$

18. $\left\{ h'_{18} := \begin{matrix} 1011 \\ 0111 \\ 1110 \\ 1101 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_5 \begin{matrix} 1110 \\ 0000 \\ 1110 \\ 1110 \end{matrix} =: g \qquad g|_{\{1,3\}} \text{ is bad}$

$g(x,y) + 4 = max_{z,w}[h'_{18}(z,w) + h'_{18}(z,y) + h'_{18}(w,x) + u_{\{1,2\}}(z) + u_{\{0\}}(w)]$

19. $\left\{ h'_{19} := \begin{matrix} 1011 \\ 1101 \\ 1010 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_5 \begin{matrix} 1010 \\ 1011 \\ 1010 \\ 1010 \end{matrix} =: g \qquad g|_{\{1,3\}} \text{ is bad}$

$g(x,y) + 4 = max_{z,w}[h'_{19}(z,w) + h'_{19}(z,y) + h'_{19}(x,z) + u_{\{2\}}(z) + u_{\{2\}}(w) + u_{\{1,3\}}(x)]$

20. $\left\{ h'_{20} := \begin{matrix} 1100 \\ 1101 \\ 1000 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_2 \begin{matrix} 1110 \\ 1101 \\ 1011 \\ 0111 \end{matrix} =: g \qquad \pi(g) = h'_{18} \text{ where } \pi(0,1,2,3) = (0,3,1,2)$

$g(x,y) + 1 = h'_{20}(x,y) + h'_{20}(y,x) + u_{\{2,3\}}(x) + u_{\{2,3\}}(y)$

21. $\left\{ h'_{21} := \begin{matrix} 1101 \\ 0110 \\ 0110 \\ 1001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_6 \begin{matrix} 1001 \\ 1101 \\ 0000 \\ 1001 \end{matrix} =: g \qquad g|_{\{0,1,2\}} \text{ is bad}$

$g(x,y) + 5 = max_{z,w}[h'_{21}(z,w) + h'_{21}(z,x) + h'_{21}(z,y) + h'_{21}(w,x) + u_{\{3\}}(z) + u_{\{0\}}(w) + u_{\{1,2\}}(x)]$

22. $\left\{ h'_{22} := \begin{matrix} 1101 \\ 1100 \\ 0010 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix} 1000 \\ 1110 \\ 0010 \\ 1010 \end{matrix} =: g \qquad \pi(g^t) = h'_9 \text{ where } \pi(0,1,2,3) = (0,2,1,3)$

$g(x,y) + 2 = max_z[h'_{22}(x,z) + h'_{22}(y,z) + u_{\{2,3\}}(z) + u_{\{1,3\}}(x)]$

23. $\left\{ h'_{23} := \begin{matrix} 1101 \\ 1110 \\ 0000 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_2 \begin{matrix} 1101 \\ 1110 \\ 0111 \\ 1011 \end{matrix} =: g \qquad \pi(g) = h'_{18} \text{ where } \pi(0,1,2,3) = (0,2,1,3)$

$g(x,y) + 1 = h'_{23}(x,y) + h'_{23}(y,x) + u_{\{2,3\}}(x) + u_{\{2,3\}}(y)$

24. $\left\{ h'_{24} := \begin{matrix} 1101 \\ 1110 \\ 0110 \\ 1001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_6 \begin{matrix} 1001 \\ 1101 \\ 0000 \\ 1001 \end{matrix} =: g \qquad g|_{\{0,1,2\}} \text{ is bad}$

$g(x,y) + 5 = max_{z,w}[h'_{24}(z,w) + h'_{24}(z,x) + h'_{24}(z,y) + h'_{24}(x,z) + u_{\{3\}}(z) + u_{\{3\}}(w) + u_{\{1,2\}}(x)]$

25. $\left\{ h'_{25} := \begin{matrix} 1110 \\ 1100 \\ 0000 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_2 \begin{matrix} 0000 \\ 1101 \\ 0001 \\ 0001 \end{matrix} =: g \qquad \pi(g^t) = h'_2$ where $\pi(0,1,2,3) = (1,3,0,2)$

$g(x,y) + 1 = h'_{25}(x,y) + u_{\{1,2,3\}}(x) + u_{\{3\}}(y)$

26. $\left\{ h'_{26} := \begin{matrix} 1110 \\ 1100 \\ 1010 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_2 \begin{matrix} 0000 \\ 1101 \\ 1011 \\ 0001 \end{matrix} =: g \qquad \pi(g) = h'_9$ where $\pi(0,1,2,3) = (1,2,3,0)$

$g(x,y) + 1 = h'_{26}(x,y) + u_{\{1,2,3\}}(x) + u_{\{3\}}(y)$

27. $\left\{ h'_{27} := \begin{matrix} 1110 \\ 1101 \\ 1010 \\ 0000 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_2 \begin{matrix} 0110 \\ 0101 \\ 0010 \\ 0111 \end{matrix} =: g \qquad \pi(g^t) = h'_{13}$ where $\pi(0,1,2,3) = (1,2,3,0)$

$g(x,y) + 1 = h'_{27}(x,y) + u_{\{3\}}(x) + u_{\{1,2,3\}}(y)$

Appendix B: Strict implementations from Case 2

The rules for reading implementations are the same as in Appendix B. Each implementation implements some predicate g such that, for some $D' \subset D$, $g|_{D'}$ is bad, or else a pair for which a strict implementation has already been found.

1. $\left\{ h := \begin{matrix} 1100 \\ 0000 \\ 0000 \\ 0001 \end{matrix}, f := \begin{matrix} 1000 \\ 0001 \\ 0000 \\ 0001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix} 1000 \\ 1101 \\ 0000 \\ 0101 \end{matrix} =: g \qquad g|_{\{0,1,2\}}$ is bad

$g(x,y) + 2 = max_z[f(x,z) + f(y,z) + h(z,x) + u_{\{1\}}(z) + u_{\{1,2\}}(x)]$

2. $\left\{ h := \begin{matrix} 1100 \\ 0000 \\ 0000 \\ 0001 \end{matrix}, f := \begin{matrix} 1110 \\ 0001 \\ 0000 \\ 0001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix} 1000 \\ 0001 \\ 0000 \\ 0001 \end{matrix} =: g \qquad (h,g)$ is Pair 1

$g(x,y) + 2 = max_z[f(x,z) + h(x,z) + h(y,z) + u_{\{2\}}(z) + u_{\{1,2\}}(x)]$

3. $\left\{ h := \begin{matrix} 1100 \\ 0000 \\ 0000 \\ 0001 \end{matrix}, f := \begin{matrix} 1000 \\ 1001 \\ 1000 \\ 0001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_4 \begin{matrix} 0101 \\ 0000 \\ 0000 \\ 0101 \end{matrix} =: g \qquad g|_{\{0,1\}}$ is bad

$g(x,y) + 3 = max_{w,z}[f(z,w) + f(y,w) + h(x,z) + u_{\{2\}}(z) + u_{\{3\}}(w)]$

4. $\left\{ h := \begin{matrix} 1100 \\ 0000 \\ 0000 \\ 0001 \end{matrix}, f := \begin{matrix} 1010 \\ 1010 \\ 1010 \\ 0001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_2 \begin{matrix} 1000 \\ 1001 \\ 1000 \\ 0001 \end{matrix} =: g \qquad (h,g)$ is Pair 3

$g(x,y) + 1 = max_z[f(z,x) + h(y,z) + u_{\{1\}}(x)]$

5. $\left\{ h := \begin{matrix} 1100 \\ 0000 \\ 0000 \\ 0001 \end{matrix}, f := \begin{matrix} 1010 \\ 1011 \\ 1010 \\ 0001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_2 \begin{matrix} 1000 \\ 1001 \\ 1000 \\ 0001 \end{matrix} =: g \qquad (h,g)$ is Pair 3

$g(x,y) + 1 = max_z[f(x,z) + h(y,z)]$

6. $\left\{ h := \begin{matrix} 1100 \\ 0000 \\ 0000 \\ 0001 \end{matrix}, f := \begin{matrix} 1110 \\ 0001 \\ 1110 \\ 0001 \end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix} 1010 \\ 1010 \\ 1010 \\ 0001 \end{matrix} =: g \qquad (h,g)$ is Pair 4

$g(x,y)+2 = max_z[f(z,x)+f(z,y)+f(y,z)]$

7. $\left\{ h := \begin{matrix}1100\\0000\\0000\\0001\end{matrix}, f := \begin{matrix}1010\\1011\\1011\\0001\end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_2 \begin{matrix}1000\\1001\\1000\\0001\end{matrix} =: g$ (h,g) is Pair 3

$g(x,y)+1 = f(y,x)+u_{\{1\}}(x)+u_{\{0,3\}}(y)$

8. $\left\{ h := \begin{matrix}1110\\0000\\0001\\0001\end{matrix}, f := \begin{matrix}1010\\0001\\0001\\0001\end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix}1000\\1101\\0101\\0101\end{matrix} =: g$ $g|_{\{0,1,2\}}$ is bad

$g(x,y)+2 = max_z[f(z,y)+f(y,z)+h(x,z)+u_{\{1\}}(x)+u_{\{1\}}(y)]$

9. $\left\{ h := \begin{matrix}1110\\0000\\0001\\0001\end{matrix}, f := \begin{matrix}1000\\0001\\1001\\0001\end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_2 \begin{matrix}1010\\0000\\0111\\0111\end{matrix} =: g$ $g|_{\{0,1,2\}}$ is bad

$g(x,y)+1 = max_z[f(y,z)+h(x,z)]$

10. $\left\{ h := \begin{matrix}1110\\0000\\0001\\0001\end{matrix}, f := \begin{matrix}1010\\0101\\0101\\0101\end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_4 \begin{matrix}1010\\0001\\0001\\0001\end{matrix} =: g$ (h,g) is Pair 8

$g(x,y)+3 = max_z[f(z,y)+f(x,z)+h(z,y)+u_{\{0,3\}}(z)]$

11. $\left\{ h := \begin{matrix}1110\\0000\\0001\\0001\end{matrix}, f := \begin{matrix}1000\\0101\\1101\\0101\end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix}1010\\0000\\0111\\0111\end{matrix} =: g$ $g|_{\{0,1,2\}}$ is bad

$g(x,y)+2 = max_z[f(y,z)+h(x,z)+u_{\{0,3\}}(z)]$

12. $\left\{ h := \begin{matrix}1110\\0000\\0001\\0001\end{matrix}, f := \begin{matrix}1000\\1101\\1101\\0101\end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_2 \begin{matrix}1000\\0001\\1001\\0001\end{matrix} =: g$ (h,g) is Pair 9

$g(x,y)+1 = f(y,x)+u_{\{2\}}(x)+u_{\{0,3\}}(y)$

13. $\left\{ h := \begin{matrix}1000\\1001\\0001\\0001\end{matrix}, f := \begin{matrix}1000\\1001\\1000\\0001\end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_4 \begin{matrix}1000\\1101\\0101\\0101\end{matrix} =: g$ $g|_{\{0,1,2\}}$ is bad

$g(x,y)+3 = max_z[f(z,y)+f(y,z)+h(x,z)+u_{\{3\}}(z)+u_{\{0,1,2\}}(y)]$

14. $\left\{ h := \begin{matrix}1000\\1001\\0001\\0001\end{matrix}, f := \begin{matrix}1010\\1011\\1010\\0001\end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix}1000\\1001\\1000\\0001\end{matrix} =: g$ (h,g) is Pair 13

$g(x,y)+2 = max_z[f(z,y)+f(x,z)+h(y,z)]$

15. $\left\{ h := \begin{matrix}1000\\1001\\0001\\0001\end{matrix}, f := \begin{matrix}1010\\1011\\1011\\0001\end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_2 \begin{matrix}1000\\1001\\1000\\0001\end{matrix} =: g$ (h,g) is Pair 13

$g(x,y)+1 = f(y,x)+u_{\{1\}}(x)+u_{\{0,3\}}(y)$

16. $\left\{ h := \begin{matrix}1100\\1101\\0001\\0001\end{matrix}, f := \begin{matrix}1010\\1101\\1010\\0000\end{matrix} \right\} \cup \mathcal{U} \overset{s}{\Longrightarrow}_3 \begin{matrix}1100\\0101\\1011\\0101\end{matrix} =: g$ $g|_{\{0,1,2\}}$ is bad

$g(x,y)+2 = max_z[f(y,x)+h(z,y)+h(x,z)+u_{\{3\}}(z)]$

17. $\left\{\begin{array}{l} h := \begin{array}{c}1100\\1101\\0001\\0001\end{array}, f := \begin{array}{c}1100\\1100\\1011\\0000\end{array}\end{array}\right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_4 \begin{array}{c}1100\\0100\\0011\\0111\end{array} =: g \qquad g|_{\{0,1,2\}}$ is bad

$g(x,y)+3 = max_z[f(x,y)+h(x,z)+h(y,z)+u_{\{3\}}(z)+u_{\{0,3\}}(x)]$

18. $\left\{\begin{array}{l} h := \begin{array}{c}1100\\1101\\0001\\0001\end{array}, f := \begin{array}{c}0000\\1101\\1011\\0000\end{array}\end{array}\right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_4 \begin{array}{c}0000\\1011\\1011\\1011\end{array} =: g \qquad g|_{\{0,1\}}$ is bad

$g(x,y)+3 = max_{w,z}[f(w,y)+h(w,z)+h(x,z)+u_{\{2\}}(w)]$

19. $\left\{\begin{array}{l} h := \begin{array}{c}1100\\1101\\0001\\0001\end{array}, f := \begin{array}{c}1010\\0001\\1011\\0001\end{array}\end{array}\right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_2 \begin{array}{c}1010\\1111\\0111\\0111\end{array} =: g \qquad g|_{\{0,1,2\}}$ is bad

$g(x,y)+1 = max_z[f(y,z)+h(x,z)]$

20. $\left\{\begin{array}{l} h := \begin{array}{c}1100\\1101\\0001\\0001\end{array}, f := \begin{array}{c}1110\\0101\\0000\\0101\end{array}\end{array}\right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_2 \begin{array}{c}1010\\0001\\1011\\0001\end{array} =: g \qquad (h,g)$ is Pair 19

$g(x,y)+1 = f(x,y)+u_{\{2\}}(x)+u_{\{0,2,3\}}(y)$

21. $\left\{\begin{array}{l} h := \begin{array}{c}1100\\1101\\0001\\0001\end{array}, f := \begin{array}{c}1010\\0101\\1010\\0101\end{array}\end{array}\right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_4 \begin{array}{c}1100\\0111\\0000\\0111\end{array} =: g \qquad g|_{\{0,1,2\}}$ is bad

$g(x,y)+3 = max_z[f(z,x)+h(x,z)+h(y,z)+u_{\{0,3\}}(z)]$

22. $\left\{\begin{array}{l} h := \begin{array}{c}1100\\1101\\0001\\0001\end{array}, f := \begin{array}{c}0000\\0101\\1011\\0101\end{array}\end{array}\right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_2 \begin{array}{c}1010\\1101\\1010\\0000\end{array} =: g \qquad (h,g)$ is Pair 16

$g(x,y)+1 = f(y,x)+u_{\{0,1,2\}}(x)+u_{\{0\}}(y)$

23. $\left\{\begin{array}{l} h := \begin{array}{c}1100\\1101\\0001\\0001\end{array}, f := \begin{array}{c}0000\\1101\\0011\\0011\end{array}\end{array}\right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_2 \begin{array}{c}1100\\1100\\1011\\0000\end{array} =: g \qquad (h,g)$ is Pair 17

$g(x,y)+1 = f(y,x)+u_{\{0,1,2\}}(x)+u_{\{0\}}(y)$

24. $\left\{\begin{array}{l} h := \begin{array}{c}1000\\1001\\1001\\0001\end{array}, f := \begin{array}{c}0000\\1101\\1011\\0000\end{array}\end{array}\right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_4 \begin{array}{c}0000\\1011\\1011\\1011\end{array} =: g \qquad g|_{\{0,1\}}$ is bad

$g(x,y)+3 = max_{w,z}[f(z,w)+f(w,y)+h(x,z)+u_{\{1,3\}}(z)+u_{\{2\}}(w)]$

25. $\left\{\begin{array}{l} h := \begin{array}{c}1100\\1100\\1101\\0001\end{array}, f := \begin{array}{c}1001\\0100\\1101\\1001\end{array}\end{array}\right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_4 \begin{array}{c}1101\\1001\\1101\\1101\end{array} =: g \qquad g|_{\{0,1\}}$ is bad

$g(x,y)+3 = max_{w,z}[f(z,y)+f(x,w)+h(z,w)+u_{\{0\}}(z)+u_{\{3\}}(w)]$

26. $\left\{ h := \begin{matrix} 1100 \\ 1100 \\ 1101 \\ 0001 \end{matrix} , f := \begin{matrix} 1101 \\ 0100 \\ 1101 \\ 1001 \end{matrix} \right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_3 \begin{matrix} 1001 \\ 0100 \\ 1101 \\ 1001 \end{matrix} =: g$ (h, g) is Pair 25

$g(x, y) + 2 = max_z[f(z, x) + f(z, y) + u_{\{1,3\}}(z) + u_{\{2\}}(x)]$

27. $\left\{ h := \begin{matrix} 1100 \\ 1100 \\ 0011 \\ 0011 \end{matrix} , f := \begin{matrix} 1001 \\ 0110 \\ 0110 \\ 1001 \end{matrix} \right\} \cup \mathcal{U} \stackrel{s}{\Longrightarrow}_4 \begin{matrix} 1111 \\ 1001 \\ 1001 \\ 1111 \end{matrix} =: g$ $g|_{\{0,1\}}$ is bad

$g(x, y) + 3 = max_{w,z}[f(z, y) + f(w, x) + h(z, w) + u_{\{3\}}(z) + u_{\{0\}}(w)]$