The approximability of three-valued Max CSP

Peter Jonsson, Mikael Klasson
Department of Computer and Information Science
University of Linköping, Sweden
peter.jonsson@ida.liu.se, mikkl@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 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. It is known that every Boolean (that is, two-valued) Max CSP problem with a finite set of allowed constraint types is either solvable exactly in polynomial time or else **APX**-complete (and hence can have no polynomial time approximation scheme unless $\mathbf{P} = \mathbf{NP}$). It has been an open problem for several years whether this result can be extended to non-Boolean Max CSP, which is much more difficult to analyze than the Boolean case. In this paper, we make the first step in this direction by establishing this result for Max CSP over a three-element domain. Moreover, we present a simple description of all polynomial-time solvable cases of our problem. This description uses the well-known algebraic combinatorial property of supermodularity. We also show that every hard three-valued Max CSP problem contains, in a certain specified sense, one of the two basic hard Max CSP problems which are the Maximum k-colourable subgraph problems for $k = 2, 3$.

Keywords: maximum constraint satisfaction, approximability, dichotomy, supermodularity.

1 Introduction and Related Work

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 [1].

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 graph theory and propositional logic. Moreover, certain CSPs were used to build foundations for the theory of complexity for optimization problems [21], and some CSPs provided material for the first optimal inapproximability results [16] (see also survey [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). 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 these problems.

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

Definition 1.2 *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, to maximize the function $f : D^n \to \mathbb{Z}^+$, defined by $f(x_1, \ldots, x_n) = \sum_{i=1}^{q} \varrho_i \cdot f_i(\mathbf{x}_i)$

Note that throughout the paper the values 0 and 1 taken by any predicate will be considered, rather unusually, as integers, not as Boolean values, and addition will always denote the addition of integers. It easy to check that, in the Boolean case, our problem coincides with the MAX CSP problem considered in [9, 10, 18]. We say that a predicate is non-trivial if it is not identically 0. Throughout the paper, we assume that $\mathcal{F}$ is finite and contains only non-trivial predicates.

Boolean constraint satisfaction problems (that is, when $D = \{0, 1\}$) are by far better studied [10] 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 [10]. 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 [10], which cannot be taken for granted due to Ladner's theorem. After the celebrated work of Schaefer [22] presenting a tractable versus **NP**-complete dichotomy for Boolean decision CSPs, many classification results have been obtained (see, e.g., [10]), most of which are dichotomies. In particular, a dichotomy in complexity and approximability for Boolean MAX CSP has been obtained by Creignou [9], and it was slightly refined in [18] (see also [10]).

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 [14]. Non-Boolean CSPs provide a much wider variety of computational problems. Moreover, research in non-Boolean CSPs leads to new sophisticated algorithms (e.g., [3]) or to new applications of known algorithms (e.g., [7]). 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 and counting problems [2, 4, 5, 8] while ideas from combinatorial optimization and operational research have been recently suggested for optimization problems [7].

The Max-CSP framework has been well-studied in the Boolean case. Many fundamental results have been obtained, concerning both complexity classifications and approximation properties (see, e.g., [9, 10, 16, 17, 18, 28]). In the non-Boolean case, a number of results have been obtained that concern exact (superpolynomial) algorithms or approximation properties (see, e.g., [11, 12, 13, 23]). The main research problem we will look at in this paper is the following.

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

It is known that, for any $\mathcal{F}$, Max CSP($\mathcal{F}$) is an **NPO** problem that belongs to the complexity class **APX**. In other words, for any $\mathcal{F}$, there is a polynomial-time approximation algorithm for Max CSP($\mathcal{F}$) whose performance is bounded by a constant.

For the Boolean case, Problem 1 was solved in [9, 10, 18]. It appears that a Boolean Max CSP($\mathcal{F}$) also exhibits a dichotomy in that it either is solvable exactly in polynomial time or else does not admit a PTAS (polynomial-time approximation scheme) unless **P**=**NP**. These papers also describe the boundary between the two cases.

In this paper we solve the above problem for the case $|D| = 3$ by showing that Max CSP($\mathcal{F}$) is solvable exactly in polynomial time if, after removing redundant values, if there are any, from the domain (that is, taking the core), all predicates in $\mathcal{F}$ are supermodular with respect to some linear ordering of the reduced domain (see definitions in Section 2.2) or else the problem is **APX**-complete. Experience shows that non-Boolean constraint problems are much more difficult to classify, and hence we believe that the techniques used in this paper can be further extended to all finite domains D. A small technical difference between our result and that of [18] is that we allow repetitions of variables in constraints, as in [10]. Similarly to [10, 18], weights do not play much role, since the tractability part of our result holds for the weighted case, while the hardness part is true in the unweighted case even if repetitions of constraints in instances are disallowed. Our result uses a combinatorial property of supermodularity which is a well-known source of tractable optimization problems [6, 15, 24], and the technique of strict implementations [10, 18] 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 [7]. Generally, it has been known for a while that the property of supermodularity allows one to solve many maximization problems in polynomial time [6, 15, 24]; however, our result is surprising in that supermodularity appears to be the *only* source of tractability for Max CSP($\mathcal{F}$). In the area of approximability, examples of other works, where hardness results are obtained for large families of problems simultaneously, include [20, 27].

The only other known complete dichotomy result on a non-Boolean constraint problem (that is, with *no* restrictions on $\mathcal{F}$) is the theorem of Bulatov [2], where the complexity of the standard decision problem CSP on a three-element domain is classified. Despite the clear similarity in the settings and also in the outcomes (full dichotomy in both cases), we note that none of the universal-algebraic techniques used in [2] can possibly be applied in the study of Max CSP because the main algebraic constructions which preserve the complexity of decision problems can be easily shown not to do this in the case of optimization problems. Another similarity between Bulatov's result and our theorem is that the proof is broken down to a (relatively) large number of cases. We believe that this is caused either by insufficiently general methods or, more likely, by significant variation in structure of the problems under consideration, where a large number of cases is probably an unavoidable feature of complete classifications.

The structure of the paper is as follows: Section 2 contains definitions of approximation complexity classes and reductions, descriptions of our reduction techniques, and the basics of supermodularity. Section 3 contains the proof of the main theorem of the paper. Finally, Section 4 contains a discussion of the work we have done and of possible future work.

2 Preliminaries

This section is subdivided into two parts. The first one contains basic definitions on complexity of approximation and our reduction techniques, while the second one introduces the notion of supermodularity and discusses the relevance of this notion in the study of Max CSP.

2.1 Approximability

2.1.1 Definitions

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* is, given an instance $\mathcal{I}$, to find a feasible solution of *optimum* value. The optimal value 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 instances and solutions can be recognized in polynomial time, solutions are polynomial-bounded in the input size, and the objective function can be computed in polynomial time (see, e.g., [1]).

Definition 2.1 (performance ratio) *A solution s to an instance $\mathcal{I}$ of an* **NPO** *problem Π is r-approximate if it has value Val satisfying*

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

where $Opt(\mathcal{I})$ is the optimal value 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.*

Completeness in **APX** is defined using an appropriate reduction, called *AP*-reduction. Our definition of this reduction follows [10, 18].

Definition 2.3 (*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*

1. *for any instance $\mathcal{I}$ of Π_1, $F(\mathcal{I})$ is an instance of Π_2;*
2. *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 $\mathcal{I}$;*

3. *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 [1]) that AP-reductions compose. It is known that Max CSP($\mathcal{F}$) belongs to **APX** for every $\mathcal{F}$ [7], and a complete classification of the complexity of Max CSP($\mathcal{F}$) for a two-element set D was obtained in [18]; we will give it in Subsection 2.2. We shall now give an example of an **APX**-complete problem which will be used extensively in this paper.

Example 2.4 *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 problem (it is Problem GT33 in [1]). Let neq_k denote the binary disequality predicate on $\{0, 1, \ldots, k-1\}$, $k \geq 2$, that is, $neq_k(x, y) = 1 \Leftrightarrow x \neq y$. The problem* Max CSP($\{neq_k\}$) *is slightly more general than the* Maximum k-colourable Subgraph *problem. To see this, think of vertices of a given graph as of variables, and apply the predicate to every pair of variables x, y such that (x, y) is an edge in the graph.*

If we allow weights on edges in graphs and on constraints then the problems are precisely the same. For unweighted problems, the Max CSP($\{neq_k\}$) *is slightly more general because one can have constraints $neq_k(x, y)$ and $neq_k(y, x)$ in the same instance. In any case, it follows that the problem* Max CSP($\{neq_k\}$) *is* **APX**-*complete.*

Interestingly, the problems Max CSP($\{neq_k\}$), $k = 2, 3$, will be the only basic hard problems for the case $|D| \leq 3$. We will show that, for all other **APX**-complete problems Max CSP($\mathcal{F}$), the set $\mathcal{F}$ can express, in a certain regular approximability-preserving way, one of the predicates neq_2, neq_3.

2.1.2 Reduction techniques

The basic reduction technique in our **APX**-completeness proofs is based on *strict implementations*, see [10, 18] where this notion was defined and used only for the Boolean case. We will give this definition in a different form from that of [10, 18], but it can easily be checked to be equivalent to the original one (in the case $|D| = 2$).

Definition 2.5 *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*

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

is satisfied 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 [10, 18].

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 $\mathcal{F} \stackrel{s}{\Longrightarrow}_\alpha f$. 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. In this paper, we will use about 60 (relatively) short strict implementations for the case when $|D| = 3$. Each of them can be straightforwardly verified by hand, or (better still) by a simple computer program[1].

Lemma 2.6 *If $\mathcal{F}$ strictly implements a predicate f, and* Max CSP$(\mathcal{F} \cup \{f\})$ *is* **APX**-*complete, then* Max CSP$(\mathcal{F})$ *is* **APX**-*complete as well.*

Proof: We need to show that Max CSP$(\mathcal{F} \cup \{f\})$ is AP-reducible to Max CSP$(\mathcal{F})$. For the case $|D| = 2$, this was proved in Lemma 5.18 of [10]. To show this for the general case, repeat the proof of the above mentioned lemma from [10], replacing 2 by $|D|$. $\Box$

Lemma 2.6 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'.

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

Example 2.7 *The* (SIMPLE) MAX CUT *problem is the problem of partitioning the set of vertices of a given undirected graph into two subsets so as to maximize the number of edges with ends being in different subsets. This problem is the same as* MAXIMUM 2-COLOURABLE SUBGRAPH *(see Example 2.4), and hence it is* **APX***-complete (see Problem ND14 in [1]). As was mentioned in Example 2.4, this problem is essentially the same as* MAX CSP($\{neq_2\}$). *Let f_{dicut} be the binary predicate on $\{0,1\}$ such that $f_{dicut}(x,y) = 1 \Leftrightarrow x = 0, y = 1$. Then* MAX CSP($\{f_{dicut}\}$) *is essentially the problem* MAX DICUT *(see problem ND16 in [1]), which is the problem of partitioning the vertices of a digraph into two subsets V_0 and V_1 so as to maximize the number of arcs going from V_0 to V_1. This problem is known to be* **APX***-complete as well, and this can be proved by exhibiting a strict 1-implementation from f_{dicut} to neq_2. Here it is: $neq_2(x,y) = f_{dicut}(x,y) + f_{dicut}(y,x)$.*

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. We will now give two more examples of strict implementations that will be used later in our proofs.

Example 2.8 *Let $D = \{0,1,2\}$, and g_i, $i = 0,1,2$, be the binary predicates on D defined by the following rule: $g_i(x,y) = 1 \Leftrightarrow (x = y = i$ or $x,y \in D \setminus \{i\})$. We will show that $\mathcal{F} = \{g_0, g_1, g_2\} \cup \mathcal{U}_D$ strictly implements the binary predicate $g(x,y)$ such that $g(x,y) = 1 \Leftrightarrow x = 0, y = 1$. Indeed, one can check that the following is a strict 5-implementation:*

$$g(x,y) + 4 = \max_{z,w}[g_0(x,z) + g_1(y,w) + g_2(z,w) + u_{\{0\}}(z) + u_{\{1,2\}}(w)].$$

Example 2.9 *In this example, we will show that the predicate neq_3 can be strictly implemented from the binary equality predicate eq_3 and all unary predicates on $D = \{0,1,2\}$. We will use three additional binary predicates f_1, f_2, f_3 defined as follows:*

$$f_1(x,y) = 1 \Leftrightarrow x \leq y,$$

$$f_2(x,y) = 1 \Leftrightarrow (x,y) = (1,2),$$

$$f_3(x,y) = 1 \Leftrightarrow (x,y) \in \{(1,0),(1,2),(2,0)\}.$$

Then it can be checked that the following equalities hold:

$$
\begin{aligned}
f_1(x,y)+3 &= \max_{z,w}[eq_3(z,w)+eq_3(z,y)+eq_3(w,x)+u_{\{2\}}(z)+ \\
&\quad +u_{\{1\}}(w)+u_{\{0\}}(x)]; \\
f_2(x,y)+5 &= \max_{z,w}[f_1(z,w)+f_1(w,y)+f_1(x,z)+u_{\{0,1\}}(z)+ \\
&\quad +u_{\{0,2\}}(w)+u_{\{1,2\}}(x)]; \\
f_3(x,y)+2 &= \max_{z,w}[f_2(z,w)+f_2(z,x)+f_2(w,z)+f_2(w,y)+f_2(x,w)+ \\
&\quad +f_2(x,y)+f_2(y,z)++u_{\{0\}}(y)]; \\
neq_3(x,y) &= f_3(x,y)+f_3(y,x).
\end{aligned}
$$

As mentioned above, a chain of strict implementations can be replaced by a single strict implementation. Since Max CSP($\{neq_3\}$) *is* **APX**-*complete by Example 2.4, Lemma 2.6 imply that the problem* Max CSP($\{eq_3\}\cup\mathcal{U}_D$) *is* **APX**-*complete as well. Note that this result was first proved in [7].*

Another notion which we will use in our hardness proofs is the notion of a core for a set of predicates. In the case when $\mathcal{F}$ consists of a single binary predicate h, this notion coincides with the usual notion of a core of the directed graph whose arcs are specified by h.

Definition 2.10 *An* endomorphism *of $\mathcal{F}$ is a unary operation π on D such that, for all $f\in\mathcal{F}$ and all $(a_1,\ldots,a_m)\in D^m$, we have $f(a_1,\ldots,a_m)=1 \Rightarrow f(\pi(a_1),\ldots,\pi(a_m))=1$. We will say that $\mathcal{F}$ is a* core *if every endomorphism of $\mathcal{F}$ is injective (i.e., a permutation).*

If π is an endomorphism of $\mathcal{F}$ with a minimal image $im(\pi)=D'$ then a core of $\mathcal{F}$, denoted $core(\mathcal{F})$, is the subset $\{f|_{D'} \mid f\in\mathcal{F}\}$ of $R_{D'}$.

The intuition here is that if $\mathcal{F}$ is not a core then it has a non-injective endomorphism π, which implies that, for every assignment φ, there is another assignment $\pi\varphi$ 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 $\mathcal{F}$. The following rather simple lemma will be frequently used in our proofs.

Lemma 2.11 *If $\mathcal{F}' = core(\mathcal{F})$ and* MAX CSP$(\mathcal{F}')$ *is* **APX**-*complete then so is* MAX CSP$(\mathcal{F})$.

Proof: We produce an AP-reduction from MAX CSP$(\mathcal{F}')$ to MAX CSP$(\mathcal{F})$. We may assume that the endomorphism $\pi : D \to D'$ is the identity on D', since if it is not, then one of its powers is such an endomorphism. We will now describe functions F and G necessary for the reduction. The function F takes an instance of MAX CSP$(\mathcal{F}')$ and replaces every predicate $f|_{D'}$ in it by f. If $\mathcal{I}$ is an instance of MAX CSP$(\mathcal{F}')$, with the set V of variables, and s' is a feasible solution of $F(\mathcal{I})$ (that is, an assignment $V \to D$) then $G(F(\mathcal{I}), s') = s$ defined by $s(x) = \pi(s'(x))$ for all $x \in V$. It is easy to see that s is also a feasible solution for $\mathcal{I}$. Finally, note that, since π is an endomorphism, s satisfies every constraint satisfied by s'; in particular, we have $Opt(\mathcal{I}) = Opt(F(\mathcal{I}))$. Hence, if s' is an r-approximate solution for $F(\mathcal{I})$ then s is an r-approximate solution for $\mathcal{I}$, so we can choose $\alpha = 1$ in the definition of AP-reducibility. $\square$

Example 2.12 *Let f be the binary predicate on $\{0,1\}$ considered in Example 2.7, and g the binary predicate on $\{0,1,2\}$ considered in Example 2.8. It is easy to see that $\{f\}$ is the core of $\{g\}$ where the corresponding endomorphism is given by $\pi(0) = 0, \pi(1) = \pi(2) = 1$. Since* MAX CSP$(\{f\})$ *is* **APX**-*complete, Lemma 2.11 implies that* MAX CSP$(\{g\})$ *is* **APX**-*complete as well. Now note that this also proves that* MAX CSP$(\{g_0, g_1, g_2\} \cup \mathcal{U}_{\{0,1,2\}})$, *as considered in Example 2.8, is* **APX**-*complete.*

2.2 Supermodularity

In this section we discuss the well-known combinatorial algebraic property of supermodularity [24] 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 2.13 *Let $\mathcal{L}$ be a lattice on D. A function $f : D^n \to \mathbb{Z}^+$ is called* supermodular *on $\mathcal{L}$ if*

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

and f is called submodular *on $\mathcal{L}$ if the inverse inequality holds.*

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 predicate 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 2.14

1) The binary equality predicate eq_3 is not supermodular on any chain on $\{0, 1, 2\}$. Take, without loss of generality, the chain $0 < 1 < 2$. Then

$$eq_3(1,1) + eq_3(0,2) = 1 \not\leq 0 = eq_3(0,1) + eq_3(1,2).$$

2) Reconsider the predicates neq_2 and f_{dicut} from Example 2.7. It is easy to check that neither of them is supermodular on any chain on $\{0, 1\}$.

3) 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 $f_{\mathbf{a}}^{\mathbf{b}}$ defined by the rules

$$\begin{array}{lcl} f_{\mathbf{a}}(x,y) = 1 & \Leftrightarrow & (x,y) \leq \mathbf{a}, \\ f^{\mathbf{b}}(x,y) = 1 & \Leftrightarrow & (x,y) \geq \mathbf{b}, \\ f_{\mathbf{a}}^{\mathbf{b}}(x,y) = 1 & \Leftrightarrow & (x,y) \leq \mathbf{a} \textit{ or } (x,y) \geq \mathbf{b}, \end{array}$$

where the order on D^2 is component-wise. It is easy to check that every predicate of one of the forms above is supermodular on the chain. Note that such predicates were considered in [7] where they were called generalized 2-monotone. We will see later in this subsection that such predicates are generic supermodular binary predicates on a chain.

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

Observation 2.15

1. *Any chain is a distributive lattice.*

$$h_1\,\begin{matrix}000\\000\\001\end{matrix}\quad h_2\,\begin{matrix}000\\000\\011\end{matrix}\quad h_3\,\begin{matrix}000\\011\\011\end{matrix}\quad h_4\,\begin{matrix}100\\000\\000\end{matrix}\quad h_5\,\begin{matrix}100\\000\\001\end{matrix}\quad h_6\,\begin{matrix}100\\000\\011\end{matrix}\quad h_7\,\begin{matrix}100\\011\\011\end{matrix}\quad h_8\,\begin{matrix}100\\100\\000\end{matrix}\quad h_9\,\begin{matrix}100\\100\\001\end{matrix}\quad h_{10}\,\begin{matrix}100\\100\\011\end{matrix}$$

$$h_{11}\,\begin{matrix}100\\101\\001\end{matrix}\quad h_{12}\,\begin{matrix}110\\110\\000\end{matrix}\quad h_{13}\,\begin{matrix}110\\110\\001\end{matrix}$$

Figure 1: A list of binary predicates on $\{0, 1, 2\}$ that are supermodular on the chain $0 < 1 < 2$. The predicates are represented by matrices, the order of indices being also $0 < 1 < 2$.

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

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

Theorem 2.16 ([7]) *If $\mathcal{F}$ is supermodular on some distributive lattice on D, then weighted* Max CSP($\mathcal{F}$) *is in* **PO**.

Given a binary predicate $f : D^2 \to \{0, 1\}$, we will often use a $|D| \times |D|$ 0/1-matrix M to represent f: $f(x, y) = 1$ if and only if $M_{xy} = 1$. Note that this matrix is essentially the table of values of the predicate. For example, some binary predicates on $D = \{0, 1, 2\}$ that are supermodular on the chain $0 < 1 < 2$ are listed in Fig. 1. Matrices for all other binary predicates that are supermodular on $0 < 1 < 2$ can be obtained from those in the list or from the trivial binary predicate by transposing matrices (which corresponds to swapping arguments in a predicate) and by replacing some all-0 rows by all-1 rows, and the same for all-0 columns (but not for both rows and columns at the same time). This can be shown by using Lemma 2.3 of [6] or by direct exhaustive (computer-assisted) search. Note that all predicates in Fig. 1 have the form described in Example 2.14(3). For example, h_2 is $f^{(2,1)}$ and h_9 is $f^{(2,2)}_{(1,0)}$.

The property of supermodularity can be used to classify the approximability of Boolean problems Max CSP($\mathcal{F}$) (though, originally the classification was obtained and stated [9, 10, 18] without using this property). It is easy to see that $\mathcal{F} \subseteq R_{\{0,1\}}$ is not a core if and only if $f(a,\ldots,a)=1$ for some $a \in \{0,1\}$ and all $f \in \mathcal{F}$, in which case MAX CSP($\mathcal{F}$) is trivial.

Theorem 2.17 ([7, 10]) *Let $D=\{0,1\}$ and $\mathcal{F} \subseteq R_D$ be a core. If $\mathcal{F}$ is supermodular on some chain on D then* MAX CSP($\mathcal{F}$) *belongs to* **PO**. *Otherwise,* MAX CSP($\mathcal{F}$) *is* **APX**-*complete.*

Remark 2.18 *It was shown in Lemma 5.37 of [10] that $\mathcal{F}$ can strictly implement neq_2 whenever* MAX CSP($\mathcal{F}$) *is* **APX**-*complete in the above theorem.*

Combining Theorem 2.17 with Lemma 2.11, we get the following corollary which will be used often in our **APX**-completeness proofs.

Corollary 2.19 *If g' is binary predicate on $\{0,1,2\}$ and $core(\{g'\})$ is $\{g\}$ where g is non-supermodular predicate on a two-element subset of D then* MAX CSP($\{g'\}$) *is* **APX**-*complete.*

Note that there are only two (up to swapping of arguments) binary predicates g on $\{0,1\}$ such that $\{g\}$ is a core: the predicates neq_2 and f_{dicut} from Example 2.7. As mentioned above, these two predicates are non-supermodular, and f_{dicut} strictly 1-implements neq_2.

3 Main result

In this section we establish a generalization of Theorem 2.17 to the case of a three-element domain. Throughout this section, let $D=\{0,1,2\}$. Note that if $\mathcal{F} \subseteq R_D$ is not a core then, by Lemma 2.11, the problem MAX CSP($\mathcal{F}$) is either trivial (if $\mathcal{F}$ has a constant endomorphism) or else reduces to a similar problem over a two-element domain, in which case Theorem 2.17 applies.

Theorem 3.1 *Let $D=\{0,1,2\}$ and $\mathcal{F} \subseteq R_D$ be a core. If $\mathcal{F}$ is supermodular on some chain on D then weighted* MAX CSP($\mathcal{F}$) *belongs to* **PO**. *Otherwise, unweighted* MAX CSP($\mathcal{F}$) *is* **APX**-*complete even if repetitions of constraints in instances are disallowed.*

Proof: The tractability part of the proof follows immediately from Theorem 2.16 (see also Observation 2.15(1)). Assume for the rest of this section that $\mathcal{F}$ is a core and it is not supermodular on any chain on D. We will show that one of neq_2, neq_3 can be obtained from $\mathcal{F}$ by using the following two operations:

1. replacing $\mathcal{F}$ by $\mathcal{F} \cup \{f\}$ where f is a predicate that can be strictly implemented from $\mathcal{F}$;
2. taking the core of a subset of $\mathcal{F}$.

By Example 2.4 and Lemmas 2.6 and 2.19, this will establish the result.

To improve readability, we divide the rest of the proof into 3 parts: in Subsection 3.1, we establish **APX**-completeness for some small sets $\mathcal{F}$ consisting of at most two binary and several unary predicates, and also for the case when $\mathcal{F}$ contains an irreflexive non-unary predicate (see definition below). Subsection 3.2 establishes the result when all unary predicates are available, and Subsection 3.3 finishes the proof. □

Remark 3.2 *Note that it can be checked in polynomial time whether a given $\mathcal{F}$ is supermodular on some chain on D, if the predicates in $\mathcal{F}$ are given by full tables of values or only by tuples on which predicates take value 1.*

3.1 Small cases and irreflexive predicates

We say that an n-ary predicate f on D is *irreflexive* if and only if $f(d, \dots, d) = 0$ for all $d \in D$. It is easy to check that any irreflexive non-trivial predicate f is not supermodular on any chain on D. For example, if f is binary and $f(a, b) = 1$ for some $a \neq b$ then $f(a, b) + f(b, a) \geq 1$, but $f(min(a, b), min(b, a)) + f(max(a, b), max(b, a)) = 0$ due to irreflexivity.

Since a predicate f is supermodular on a chain C if and only if f is supermodular on its dual, we can identify chains on the three-element set D with the same middle element: let C_i denote an arbitrary chain on D with i as its middle element. We also define the set $\mathcal{Q}_i$ that consists of all binary predicates on D that are supermodular on C_i but on neither of the other two chains. For example, it is easy to check using Fig. 1 that $\mathcal{Q}_1$ consists of predicates $h_2, h_5, h_6, h_8, h_9, h_{10}, h_{11}$ and the predicates obtained from them by using the following operations:

1. swapping the variables (this corresponds to transposing the tables);
2. adding a unary predicate $u(x)$ or $u(y)$ in such a way that the sum remains to be a predicate (this corresponds to replacing all-0 rows/columns with all-1 rows/columns).

Recall that, for a subset $D' \subseteq D$, $u_{D'}$ denotes the predicate such that $u_{D'}(x) = 1$ if and only if $x \in D'$, and $\mathcal{U}_D = \{u_{D'} \mid \varnothing \neq D' \subseteq D\}$, that is, $\mathcal{U}_D$ is the set of all non-trivial unary predicates on D.

Lemma 3.3 *Let g be a binary predicate such that, for some $a \in D$, $g(x, a) = 1$ for all $x \in D$. Let $g'(x, y) = 0$ if $y = a$ and $g'(x, y) = g(x, y)$ otherwise. Then, the following holds:*

1. *for any chain on D, g and g' are supermodular (or not) on it simultaneously; and*
2. *$\{g, u_{D\setminus\{a\}}\})$ strictly implements g'.*

Proof: The first statement is a straightforward consequence of the definition. To see that the second statement holds, we note that $g'(x, y) + 1 = g(x, y) + u_{D\setminus\{a\}}(y)$ is a strict 2-implementation of $g'(x, y)$. $\square$

We say that a predicate g contains an *all-one column* if there exists $a \in D$ such that $g(x, a) = 1$ for all $x \in D$, and we define *all-one rows* analogously. Clearly, the lemma above holds for both all-one rows and all-one columns. The lemma will be used in our hardness proofs as follows: if $\mathcal{F}$ contains $g(x, y)$ and $u_{D\setminus\{a\}}(y)$ then, by Lemma 2.6, we may also assume that $g' \in \mathcal{F}$.

The following lemma contains more **APX**-completeness results for some problems Max CSP($\mathcal{F}$) where $\mathcal{F}$ is a small set containing at most two binary and some unary predicates.

Lemma 3.4 *Let f, h be binary predicates on D. The problem* Max CSP($\mathcal{F}$) *is* **APX***-complete if one of the following holds:*

1. *$\mathcal{F} = \{f\}$ and f is nontrivial and irreflexive;*
2. *$\mathcal{F} = \{f\} \cup \mathcal{U}_D$ and f is not supermodular on any chain on D;*
3. *$\mathcal{F} = \{f, h_7\} \cup \mathcal{U}_D$ where $f \in \mathcal{Q}_0$ and h_7 is given in Fig. 1;*
4. *$\mathcal{F} = \{f, h\} \cup \mathcal{U}_D$ and $f \in \mathcal{Q}_1$ and $h \in \mathcal{Q}_0$;*
5. *$\mathcal{F} = \{f, u_{\{0,1\}}\}$ where f is such that $f(0, 0) = f(1, 1) = 0$ and $f(2, 2) = f(0, 1) = 1$.*

Proof: The lemma is proved by providing computer-generated strict implementations, from $\mathcal{F}$, of the predicate neq_3 (see Example 2.4) or of a binary predicate whose core is a non-supermodular predicate on a two-element subset of D (see Corollary 2.19). In total, we give 54 implementations.

We prove only case 1 here; the other cases are similar and can be found in the Appendix. First, we make the list of predicates we need to consider.

There are 63 irreflexive non-trivial predicates on D. We may skip all predicates whose core is a non-supermodular predicate on a two-element subset of D, since we already have the result for them (Corollary 2.19). For every pair of predicates that can be obtained from each other by swapping the variables (that is, $f(x,y)$ and $f'(x,y) = f(y,x)$), we can skip one of them. By symmetry, we may skip any predicate obtained from some predicate already in the list by renaming the elements of D. Finally, we already know that the result is true for the disequality predicate neq_3, so we skip that one too. All this can be done using a computer or by hand, and the resulting list contains only six predicates. Here are strict implementations for them.

1. $f_1 := \begin{matrix} 011 \\ 001 \\ 000 \end{matrix} \overset{s}{\Longrightarrow}_1 \begin{matrix} 011 \\ 101 \\ 110 \end{matrix} = neq_3$

   Implementation: $neq_3 = f_1(x,y) + f_1(y,x)$

2. $f_2 := \begin{matrix} 010 \\ 001 \\ 100 \end{matrix} \overset{s}{\Longrightarrow}_1 \begin{matrix} 011 \\ 101 \\ 110 \end{matrix} = neq_3$

   Implementation: $neq_3(x,y) = f_2(x,y) + f_2(y,x)$

3. $f_3 := \begin{matrix} 011 \\ 101 \\ 100 \end{matrix} \overset{s}{\Longrightarrow}_3 \begin{matrix} 010 \\ 001 \\ 100 \end{matrix} = f_2$

   Implementation: $f_2(x,y) + 2 = \max_z[f_3(z,x) + f_3(x,y) + f_3(y,z)]$

4. $f_4 := \begin{matrix} 011 \\ 101 \\ 000 \end{matrix} \overset{s}{\Longrightarrow}_3 \begin{matrix} 011 \\ 101 \\ 110 \end{matrix} = neq_3$

   Implementation:
   $neq_3(x,y) + 2 = \max_z[f_4(z,x) + f_4(z,y) + f_4(x,y) + f_4(y,x)]$

5. $f_5 := \begin{matrix} 001 \\ 100 \\ 000 \end{matrix} \overset{s}{\Longrightarrow}_3 \begin{matrix} 011 \\ 101 \\ 000 \end{matrix} = f_4$

   Implementation:
   $f_4(x,y) + 2 = \max_{z,w}[f_5(z,w) + f_5(z,y) + f_5(w,y) + f_5(x,w) + f_5(y,z)]$

6. $f_6 := \begin{matrix} 011 \\ 001 \\ 100 \end{matrix} \overset{s}{\Longrightarrow}_4 \begin{matrix} 011 \\ 101 \\ 100 \end{matrix} = f_3$

Implementation:
$f_3(x,y)+3 = \max_{z,w}[f_6(z,y)+f_6(w,z)+f_6(w,x)+f_6(x,z)+f_6(x,y)]$

□

Proposition 3.5 *If $h \in R_D^{(n)}$, $n \geq 2$, is nontrivial and irreflexive, then* Max CSP$(\{h\})$ *is* **APX***-complete.*

Proof: The proof is by induction on n (the arity of h). The basis when $n = 2$ was proved in Lemma 3.4(1). Assume that the lemma holds for $n = k$, $k \geq 2$. We show that it holds for $n = k+1$. Assume first that there exists $(a_1,\ldots,a_{k+1}) \in D^{k+1}$ such that $h(a_1,\ldots,a_{k+1}) = 1$ and $|\{a_1,\ldots,a_{k+1}\}| \leq k$. We assume without loss of generality that $a_k = a_{k+1}$ and consider the predicate $h'(x_1,\ldots,x_k) = h(x_1,\ldots,x_k,x_k)$. Note that this is a strict 1-implementation of h', that $h'(d,\ldots,d) = 0$ for all $d \in D$, and that h' is nontrivial since $h'(a_1,\ldots,a_k) = 1$. Consequently, Max CSP$(\{h'\})$ is **APX**-complete by the induction hypothesis, and Max CSP$(\{h\})$ is **APX**-complete, too.

Assume now that $|\{a_1,\ldots,a_{k+1}\}| = k+1$ whenever $h(a_1,\ldots,a_{k+1}) = 1$. Consider the predicate $h'(x_1,\ldots,x_k) = \max_y h(x_1,\ldots,x_k,y)$, and note that this is a strict 1-implementation of h'. We see that $h'(d,\ldots,d) = 0$ for all $d \in D$ (due to the condition above) and h' is non-trivial since h is non-trivial. We can once again apply the induction hypothesis and draw the conclusion that Max CSP$(\{h'\})$ and Max CSP$(\{h\})$ are **APX**-complete. □

3.2 When all unary predicates are available

As the next step, we will prove that Max CSP$(\mathcal{F} \cup \mathcal{U}_D)$ is **APX**-complete if $\mathcal{F}$ is not supermodular on any chain. As a special case of Lemma 6.3 of [6], we have the following result (see also Observation 6.1 of [6]).

Lemma 3.6 *An n-ary, $n \geq 2$, predicate f is supermodular on a fixed chain C if and only if the following holds: every binary predicate obtained from f by replacing any given $n-2$ variables by any constants is supermodular on C.*

Proposition 3.7 Max CSP$(\mathcal{F} \cup \mathcal{U}_D)$ *is* **APX***-complete if $\mathcal{F}$ is not supermodular on any chain.*

Proof: By our initial assumptions, $\mathcal{F}$ is not supermodular on any chain. For $i = 0, 1, 2$, let $f_i \in \mathcal{F}$ be *not* supermodular on C_i. Recall that every unary predicate is supermodular on any chain. Therefore, f_i is n-ary where $n \geq 2$ (note that n depends on i). By Lemma 3.6, it is possible to substitute constants for some $n-2$ variables of f_i to obtain a binary predicate f_i' which is not supermodular on C_i. 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, $f_i'(x, y) = f_i(x, y, d_3, \ldots, d_n)$. Then the following is a strict $(n-1)$-implementation of f_i':

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

By Lemma 2.6, it now is sufficient to show the result for $\mathcal{F}$ consisting of at most three binary predicates. We can assume that $\mathcal{F}$ is minimal with the property of not being supermodular on any chain. In addition, we can assume that the binary predicates in $\mathcal{F}$ do not contain any all-one column or all-one row (this is justified by Lemma 3.3). We need to consider three cases depending on the number of predicates in $\mathcal{F}$.

Case 1 $|\mathcal{F}| = 1$.
The result is proved in Lemma 3.4(1-2).

Case 2 $|\mathcal{F}| = 2$.
Assume $\mathcal{F} = \{g, h\}$. We consider two subcases:

1. g is supermodular on C_1 and C_2 but not on C_0; this implies that $h \in \mathcal{Q}_0$ because otherwise $\mathcal{F} \cup \mathcal{U}_D$ is supermodular on C_1 or C_2, or else h is not supermodular on any chain, contradicting the minimality of $\mathcal{F}$. By Lemma 3.3, we can assume that neither g nor h have an all-1 row or column. It can be easily checked by inspecting the list of binary predicates (see, e.g., Fig. 1) that there exist only three such predicates g. These are predicates h_3, h_4 and h_7 from Fig. 1. We have that $h_4(x, y) + 1 = h_3(x, y) + u_{\{0\}}(x) + u_{\{0\}}(y)$ is a strict 2-implementation of h_4 from h_3, $h_3(x, y) + 1 = h_4(x, y) + u_{\{1,2\}}(x) + u_{\{1,2\}}(y)$ is a strict 2-implementation of h_3 from h_4, and $h_7(x, y) = h_3(x, y) + h_4(x, y)$ is a strict 1-implementation of h_7. Hence, since all unary predicates are available, it is enough to show the result for $g = h_7$, which has already been obtained in Lemma 3.4(3).

2. None of the predicates g, h is supermodular on two distinct (that is, not mutually dual) chains. By symmetry, we may assume that $g \in \mathcal{Q}_1$

and $h \in \mathcal{Q}_0$. Then the result follows from Lemma 3.4(4).

Case 3 $|\mathcal{F}| = 3$.
By the minimality of $\mathcal{F}$, it follows that $\mathcal{F} = \{g_0, g_1, g_2\}$ where each g_i is *not* supermodular on C_i, but is supermodular on the other two chains. As argued in the previous case, we may assume that $g_0 = h_7$. By symmetry, we may assume that g_1 and g_2 have the following matrices, respectively:

$$\begin{array}{ccc} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{array} \quad \text{and} \quad \begin{array}{ccc} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{array}.$$

It remains to say that, for such $\mathcal{F}$, **APX**-completeness of Max CSP$(\mathcal{F} \cup \mathcal{U}_D)$ was shown in Example 2.8. □

3.3 The Last Step

We will need one more auxiliary lemma. Let $\mathcal{C}_D = \{u_{\{d\}} \mid d \in D\}$.

Lemma 3.8 *For any $\mathcal{F}$, if* Max CSP$(\mathcal{F} \cup \mathcal{U}_D)$ *is* **APX**-*complete, then so is* Max CSP$(\mathcal{F} \cup \mathcal{C}_D)$.

Proof: For any disjoint subsets S, T of D, $u_{S \cup T}(x) = u_S(x) + u_T(x)$ is a strict 1-implementation of $u_{S \cup T}$. Use this repeatedly and apply Lemma 2.6. □

Proposition 3.9 *If $\mathcal{F}$ is not supermodular on any chain, then* Max CSP$(\mathcal{F})$ *is* **APX**-*complete.*

Proof: If $\mathcal{F}$ contains a non-trivial irreflexive predicate then the result follows from Proposition 3.5. Letting $r(f) = \{d \in D \mid f(d, \ldots, d) = 1\}$ for a predicate f, we can now assume that $r(f) \neq \varnothing$ for all $f \in \mathcal{F}$. Let $r(\mathcal{F}) = \{r(f) \mid f \in \mathcal{F}\}$. If $r(f) = S$ then $u_S(x) = f(x, \ldots, x)$ is a strict 1-implementation of $u_S(x)$. Hence, for each $S \in r(\mathcal{F})$, we can without loss of generality assume that $u_S \in \mathcal{F}$. Note that if, for some $d \in D$, we have $d \in r(f)$ for all $f \in \mathcal{F}$, then the operation sending all elements of D to d is an endomorphism of $\mathcal{F}$, contradicting the assumption that $\mathcal{F}$ is a core. Hence, for every $d \in D$, there is a unary predicate $u_S \in \mathcal{F}$ (depending on d) such that $d \notin S$.

Note that if $\{a,b,c\} = D$ then $u_{\{b\}}(x) + 1 = u_{\{a,b\}}(x) + u_{\{b,c\}}(x)$ is a strict 2-implementation of $u_{\{b\}}(x)$. Hence, we may assume that, for any distinct two-element sets S_1, S_2 in $r(\mathcal{F})$, we also have $S_1 \cap S_2 \in r(\mathcal{F})$. It is easy to see that then $r(\mathcal{F})$ contains at least one of the following: 1) two distinct singletons, or 2) sets $\{a,b\}$ and $\{c\}$ such that $\{a,b,c\} = D$. We will consider these two cases separately.

Note that, by Proposition 3.7, Max CSP$(\mathcal{F} \cup \mathcal{U}_D)$ is **APX**-complete. Then, by Lemma 3.8, Max CSP$(\mathcal{F} \cup \mathcal{C}_D)$ is **APX**-complete as well. Hence, by Lemma 2.6, showing that $\mathcal{F}$ can strictly implement every predicate in $\mathcal{C}_D$ is sufficient to prove the proposition.

Case 1 $u_{\{a\}}, u_{\{b\}} \in \mathcal{F}$ and $a \neq b$.
Assume without loss of generality that $a = 0$ and $b = 1$. We will show that $\mathcal{F}$ can strictly implement $u_{\{2\}}$. Since $\mathcal{F}$ is a core, let $f_1 \in \mathcal{F}$ be an n-ary predicate witnessing that the operation π_1 such that $\pi_1(0) = 0$ and $\pi_1(1) = \pi_1(2) = 1$ is not an endomorphism of $\mathcal{F}$. Let $\mathbf{a} = (a_1, \ldots, a_n)$ be a tuple such that $f(\mathbf{a}) = 1$, but $f(\pi_1(\mathbf{a})) = 0$ (where $\pi_1(\mathbf{a}) = (\pi_1(a_1), \ldots, \pi_1(a_n))$). Note that at least one of the a_i's must be equal to 2, since otherwise $\mathbf{a} = \pi_1(\mathbf{a})$. For each $1 \leq i \leq n$, let t_i be x if $a_i = 2$ and z_i otherwise. Denote by l the number of t_i's that are of the form z_i. Now it is not difficult to verify that

$$g_1(x) + l = \max_{\{z_i | a_i \neq 2\}} [f_1(t_1, \ldots, t_n) + \sum_{a_i \neq 2} u_{\{a_i\}}(z_i)]$$

is a strict $(l+1)$-implementation of a unary predicate $g_1(x)$ such that $g_1(2) = 1$ and $g_1(1) = 0$. That is, g_1 is either $u_{\{2\}}$ or $u_{\{0,2\}}$. If $g_1 = u_{\{2\}}$ then we have all predicates from $\mathcal{C}_D$, and we are done. So assume that $g_1 = u_{\{0,2\}}$.

Next, the operation π_2 such that $\pi_2(1) = 1$ and $\pi_2(0) = \pi_2(2) = 0$ is not an endomorphism of $\mathcal{F}$ either. Acting as above, one can show that $\mathcal{F}$ strictly implements a unary predicate $g_2(x)$ such that $g(2) = 1$ and $g(0) = 0$, which is either $u_{\{2\}}$ or $u_{\{1,2\}}$. Again, if $g_2 = u_{\{2\}}$ then we are done. Otherwise, $g_2 = u_{\{1,2\}}$ and $u_{\{2\}}(x) + 1 = g_1(x) + g_2(x)$ is strict 2-implementation of $u_{\{2\}}$.

Case 2. $u_{\{a,b\}}, u_{\{c\}} \in \mathcal{F}$ and $\{a,b,c\} = D$.
Assume without loss of generality that $a = 0$, $b = 1$ and $c = 2$. Let $f \in \mathcal{F}$ be a predicate witnessing that the operation π such that $\pi(0) = \pi(1) = 1$ and $\pi(2) = 2$ is not an endomorphism of $\mathcal{F}$. If f is unary then $f = u_{\{0\}}$ or $f = u_{\{0,2\}}$. In the former case we go back to Case 1, and in the latter case $u_{\{0\}}(x) + 1 = u_{\{0,1\}}(x) + u_{\{0,2\}}(x)$ is a strict 2-implementation of $u_{\{0\}}$, so we can use Case 1 again.

Assume that f is n-ary, $n \geq 2$. Similarly to Case 1, let $\mathbf{a} = (a_1, \ldots, a_n)$ be a tuple such that $f(\mathbf{a}) = 1$, but $f(\pi(\mathbf{a})) = 0$. For each $1 \leq i \leq n$, let t_i be x if $a_i = 0$, y if $a_i = 1$ and z otherwise. Note that y or/and z may not appear among the t_i's (unlike x which does appear). We consider the case when z does appear, the other case is very similar. If none of the t_i's is y then $g_1(x) = \max_z[f_1(t_1, \ldots, t_n) + u_{\{2\}}(z)]$ is a strict 2-implementation of a unary predicate g_1 which is either $u_{\{0\}}$ or $u_{\{0,2\}}$ (since π is not its endomorphism). Hence we are done, as above. Assume now that some t_i is y. Now it is not difficult to verify that $g_2(x, y) = \max_z[f_1(t_1, \ldots, t_n) + u_{\{2\}}(z)]$ is a strict 2-implementation of a binary predicate g_2 which satisfies $g_2(0, 1) = 1$ and $g_2(1, 1) = 0$. If $g_2(0, 0) = 1$, then the predicate $g_2(x, x)$ is either $u_{\{0\}}$ or $u_{\{0,2\}}$, and we are done. Otherwise, we have $g_2(0, 0) = 0$. Now apply Lemma 3.4(1) if $g_2(2, 2) = 0$, and use Lemma 3.4(5) otherwise. $\square$

4 Conclusion

We have proved a dichotomy result for maximum constraint satisfaction problems over a three-element domain. The property of supermodularity appears to be the dividing line: those sets of predicates whose cores have this property give rise to problems solvable exactly in polynomial time, while all other sets of predicates can implement, in a regular way, the disequality predicate on a two- or three-element set, and hence give rise to **APX**-complete problems. Interestingly, the description of polynomial cases is based on orderings of the domain, which is not suggested in any way by the formulation of the problem.

It can be shown using Theorem 2.16 that Theorem 3.1, as stated in the paper, does not hold for domains with at least four elements. The reason is that all lattices on at most three-element set are chains, but on larger sets there are other types of lattices (for example, a Boolean lattice on a four-element set). Corollary 1 of [19] implies the existence of sets $\mathcal{F}$ such that Max CSP($\mathcal{F}$) is tractable, and $\mathcal{F}$ is supermodular on some distributive lattice which is not a chain, but not supermodular on any chain. Hence, more general lattices are required to make further progress in classifying the complexity of Max CSPs, as is a better understanding of the supermodularity property on arbitrary lattices. We believe that the ideas from this paper can be further developed to obtain a complete classification of approximability of Max CSP.

Notably, the hard problems of the form considered in this paper do not have a PTAS. It is possible that, as it is done in Theorem 8.8 of [10], certain restrictions on the incidence graph of variables in the instance can give rise to **NP**-hard problems that do have a PTAS.

Finally, techniques of [26] can be used to obtain better implementations and more precise (in)approximability results for non-Boolean problems Max CSP. We leave this direction for future research.

Acknowledgements

The authors wish to thank Peter Jeavons and Benoit Larose for providing detailed comments on this paper. Peter Jonsson and Mikael Klasson were supported by the *Swedish Research Council* (VR) under grant 621–2003–3421, and Peter Jonsson was also supported by the *Center for Industrial Information Technology* (CENIIT) under grant 04.01. Andrei Krokhin was supported by UK EPSRC grant GR/T05325/01.

References

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

[2] A. Bulatov. A dichotomy theorem for constraints on a 3-element set. In *FOCS'02*, pages 649–658, 2002.

[3] A. Bulatov. Mal'tsev constraints are tractable. Technical Report RR-02-05, Computing Laboratory, Oxford University, 2002.

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

[5] A. Bulatov and V. Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. In *FOCS'03*, pages 562–571, 2003.

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

[7] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. Identifying efficiently solvable cases of Max CSP. In *STACS'04*, volume 2996 of *LNCS*, pages 152–163, 2004.

[8] D. Cohen, P. Jeavons, and M. Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997.

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

[10] 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.

[11] 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.

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

[13] L. Engebretsen and V. Guruswami. Is constraint satisfaction over two variables always easy? *Random Structures and Algorithms*, 25(2):150–178, 2004.

[14] 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.

[15] S. Fujishige. *Submodular Functions and Optimization*, volume 47 of *Annals of Discrete Mathematics*. North-Holland, 1991.

[16] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48:798–859, 2001.

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

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

[19] A. Krokhin and B. Larose. Maximum constraint satisfaction on diamonds. Technical Report CS-RR-408, Department of Computer Science, University of Warwick, UK, 2004. Submitted for publication.

[20] C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. In *ICALP'93*, volume 700 of *LNCS*, pages 40–51, 1993.

[21] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

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

[23] M. Serna, L. Trevisan, and F. Xhafa. The (parallel) approximability of non-boolean satisfiability problems and restricted integer programming. In *Proceedings STACS'98*, volume 1373 of *LNCS*, pages 488–498, 1998.

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

[25] L. Trevisan. Inapproxiambility of combinatorial optimization problems. Technical Report 065, Electronic Colloquium on Computational Complexity, 2004.

[26] L. Trevisan, G.B. Sorkin, M. Sudan, and D.P. Williamson. Gadgets, approximation, and linear programming. *SIAM Journal on Computing*, 29:2074–2097, 2000.

[27] D. Zuckerman. On unapproximable versions of NP-complete problems. *SIAM Journal on Computing*, 25:1293–1304, 1996.

[28] U. Zwick. Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint. In *Proceedings 9th ACM Symposium on Discrete Algorithms, SODA'98*, pages 201–210, 1998.

Appendix

Proof of Lemma 3.4 (Cases 2-5)

In each case, we generate a list of all applicable predicates, and then optimise it as follows:

- skip a predicate $f(x, y)$ if $f'(x, y)$ is in the list, with $f(x, y) = f'(y, x)$;
- skip all predicates with an all-1 row or column.

By Lemma 3.3, it is sufficient to prove the result for the optimised lists.

Cases 2-5 follow in order, with a description and a list of strict implementations. Each strict implementation produces either neq_3, or some binary predicate whose core is a non-supermodular predicate on a two-element subset of D, or some other predicate for which an implementation has already been found.

Cases 2 and 3 will also use eq_3 – the binary equality predicate – as implementation target. Note that it was shown in Example 2.9 that the set $\{eq_3\} \cup \mathcal{U}_D$ can strictly implement neq_3, and hence, Max CSP($\{eq_3\} \cup \mathcal{U}_D$) is **APX**-complete.

If some implementation produces a predicate g, whose core is a non-supermodular predicate on a two-element subset of D, then we write $([0, 1, 2] \mapsto [\pi(0), \pi(1), \pi(2)])$ to describe the endomorphism π leading to the core, and we also give a matrix for that non-supermodular predicate.

In all strict implementations in this section, the variables x, y are primary, and z, w (when they appear) are auxiliary.

Case 2

$\mathcal{F} = \{f\} \cup \mathcal{U}_D$ and f is not supermodular on any chain on D.

We further optimise the list of predicates for this case. We can assume that $f(d, d) = 1$ for some d, as the other predicates are handled in Case 1. By symmetry, we may skip a predicate if there is another predicate in the list by renaming the elements of D. We can also skip eq_3, as we have handled it separately in Example 2.9.

$$\left\{ f_1 := \begin{matrix} 110 \\ 010 \\ 000 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_2 \begin{matrix} 100 \\ 010 \\ 001 \end{matrix} =: g \qquad g = eq_3$$

Implementation: $g(x, y) + 1 = f_1(x, y) + f_1(y, x) + u_{\{2\}}(x) + u_{\{2\}}(y)$

$$\left\{ f_2 := \begin{matrix} 101 \\ 101 \\ 000 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_2 \begin{matrix} 000 \\ 000 \\ 010 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 00 \\ 10 \end{matrix} \ ([0,1,2] \mapsto [0,0,1])$$

Implementation: $g(x,y) + 1 = f_2(x,y) + u_{\{2\}}(x) + u_{\{1\}}(y)$

$$\left\{ f_3 := \begin{matrix} 110 \\ 001 \\ 100 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_3 \begin{matrix} 000 \\ 001 \\ 010 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 01 \\ 10 \end{matrix} \ ([0,1,2] \mapsto [0,1,0])$$

Implementation: $g(x,y) + 2 = f_3(x,y) + f_3(y,x) + u_{\{1,2\}}(x) + u_{\{1,2\}}(y)$

$$\left\{ f_4 := \begin{matrix} 110 \\ 011 \\ 101 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_2 \begin{matrix} 100 \\ 010 \\ 001 \end{matrix} =: g \qquad g = eq_3$$

Implementation: $g(x,y) + 1 = f_4(x,y) + f_4(y,x)$

$$\left\{ f_5 := \begin{matrix} 101 \\ 100 \\ 000 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_3 \begin{matrix} 000 \\ 101 \\ 000 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 00 \\ 10 \end{matrix} \ ([0,1,2] \mapsto [0,1,0])$$

Implementation: $g(x,y) + 2 = max_z[f_5(z,y) + f_5(x,z) + u_{\{2\}}(z) + u_{\{1,2\}}(x)]$

$$\left\{ f_6 := \begin{matrix} 001 \\ 010 \\ 000 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_2 \begin{matrix} 110 \\ 010 \\ 000 \end{matrix} =: g \qquad g = f_1$$

Implementation: $g(x,y) + 1 = max_z[f_6(z,x) + f_6(y,z) + u_{\{0\}}(x)]$

$$\left\{ f_7 := \begin{matrix} 011 \\ 010 \\ 000 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_3 \begin{matrix} 110 \\ 010 \\ 000 \end{matrix} =: g \qquad g = f_1$$

Implementation:
$g(x,y) + 2 = max_z[f_7(z,x) + f_7(z,y) + f_7(y,z) + u_{\{2\}}(z) + u_{\{0\}}(x)]$

$$\left\{ f_8 := \begin{matrix} 001 \\ 110 \\ 000 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_4 \begin{matrix} 001 \\ 000 \\ 000 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 01 \\ 00 \end{matrix} \ ([0,1,2] \mapsto [0,0,1])$$

Implementation:
$g(x,y) + 3 = max_z[f_8(z,y) + f_8(x,z) + u_{\{1,2\}}(z) + u_{\{0,2\}}(x) + u_{\{2\}}(y)]$

$$\left\{ f_9 := \begin{matrix} 101 \\ 110 \\ 000 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_4 \begin{matrix} 001 \\ 000 \\ 000 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 01 \\ 00 \end{matrix} \ ([0,1,2] \mapsto [0,0,1])$$

Implementation:
$g(x,y) + 3 = max_z[f_9(z,y) + f_9(x,z) + u_{\{1\}}(z) + u_{\{0,2\}}(x) + u_{\{2\}}(y)]$

$$\left\{ f_{10} := \begin{matrix} 011 \\ 110 \\ 000 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_4 \begin{matrix} 001 \\ 010 \\ 000 \end{matrix} =: g \qquad g = f_6$$

Implementation:
$g(x,y) + 3 = max_z[f_{10}(x,z) + f_{10}(x,y) + f_{10}(y,z) + u_{\{0,2\}}(z) + u_{\{2\}}(x) + u_{\{2\}}(y)]$

$$\left\{ f_{11} := \begin{matrix} 001 \\ 010 \\ 100 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_2 \begin{matrix} 100 \\ 010 \\ 001 \end{matrix} =: g \qquad g = eq_3$$

Implementation: $g(x,y) + 1 = max_z[f_{11}(z,x) + f_{11}(z,y)]$

$$\left\{ f_{12} := \begin{matrix} 011 \\ 010 \\ 100 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_3 \begin{matrix} 110 \\ 010 \\ 000 \end{matrix} =: g \qquad g = f_1$$

Implementation: $g(x,y) + 2 = max_z[f_{12}(z,y) + f_{12}(x,z) + u_{\{1,2\}}(z)]$

$$\left\{ f_{13} := \begin{matrix} 011 \\ 110 \\ 100 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_4 \begin{matrix} 011 \\ 010 \\ 000 \end{matrix} =: g \qquad g = f_7$$

Implementation:
$g(x,y) + 3 = max_z[f_{13}(z,x) + f_{13}(z,y) + f_{13}(x,y) + u_{\{0\}}(z) + u_{\{0\}}(x)]$

$$\left\{ f_{14} := \begin{matrix} 110 \\ 011 \\ 100 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_3 \begin{matrix} 000 \\ 000 \\ 110 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 00 \\ 10 \end{matrix} \ ([0,1,2] \mapsto [0,0,1])$$

Implementation: $g(x,y) + 2 = max_z[f_{14}(z,y) + f_{14}(x,z) + u_{\{2\}}(x)]$

$$\left\{ f_{15} := \begin{matrix} 101 \\ 011 \\ 100 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_4 \begin{matrix} 110 \\ 010 \\ 000 \end{matrix} =: g \qquad g = f_1$$

Implementation:
$g(x,y) + 3 = max_z[f_{15}(z,x) + f_{15}(z,y) + f_{15}(x,z) + u_{\{2\}}(z) + u_{\{1\}}(y)]$

$$\left\{ f_{16} := \begin{matrix} 011 \\ 011 \\ 100 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_4 \begin{matrix} 000 \\ 000 \\ 100 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 00 \\ 10 \end{matrix} \ ([0,1,2] \mapsto [0,0,1])$$

Implementation:
$g(x,y) + 3 = max_z[f_{16}(z,y) + f_{16}(x,z) + f_{16}(x,y) + u_{\{2\}}(z) + u_{\{2\}}(x)]$

$$\left\{ f_{17} := \begin{matrix} 110 \\ 010 \\ 001 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_3 \begin{matrix} 110 \\ 010 \\ 000 \end{matrix} =: g \qquad g = f_1$$

Implementation: $g(x,y) + 2 = max_z[f_{17}(z,y) + f_{17}(x,z) + u_{\{0,1\}}(z)]$

$$\left\{ f_{18} := \begin{matrix} 101 \\ 110 \\ 001 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_3 \begin{matrix} 110 \\ 010 \\ 000 \end{matrix} =: g \qquad g = f_1$$

Implementation: $g(x,y) + 2 = max_z[f_{18}(z,x) + f_{18}(y,z) + u_{\{0,1\}}(x)]$

$$\left\{ f_{19} := \begin{matrix} 110 \\ 100 \\ 000 \end{matrix} \right\} \cup \mathcal{U}_D \overset{s}{\Longrightarrow}_5 \begin{matrix} 000 \\ 101 \\ 000 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 00 \\ 10 \end{matrix} \ ([0,1,2] \mapsto [0,1,0])$$

Implementation: $g(x,y) + 4 =$
$max_{z,w}[f_{19}(z,w) + f_{19}(z,y) + f_{19}(w,x) + u_{\{1\}}(z) + u_{\{1\}}(w) + u_{\{1,2\}}(x) + u_{\{2\}}(y)]$

$$\left\{ f_{20} := \begin{array}{c} 101 \\ 010 \\ 100 \end{array} \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_5 \begin{array}{c} 001 \\ 010 \\ 000 \end{array} =: g \qquad g = f_6$$

Implementation:
$g(x,y)+4 = max_{z,w}[f_{20}(z,w)+f_{20}(z,y)+f_{20}(w,x)+u_{\{2\}}(z)+u_{\{1,2\}}(w)+u_{\{1,2\}}(y)]$

$$\left\{ f_{21} := \begin{array}{c} 101 \\ 011 \\ 110 \end{array} \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_4 \begin{array}{c} 101 \\ 010 \\ 100 \end{array} =: g \qquad g = f_{20}$$

Implementation: $g(x,y) + 3 = max_z[f_{21}(z,x) + f_{21}(z,y) + f_{21}(x,y) + u_{\{0,2\}}(z)]$

Case 3

$\mathcal{F} = \{f, h_7\} \cup \mathcal{U}_D$ and $f \in \mathcal{Q}_0$.

We further optimise the list of predicates for this case as follows. By symmetry, we can skip a predicate if there is another predicate in the list obtained by swapping the names of elements 1 and 2 of D.

$$\left\{ f_1 := \begin{array}{c} 000 \\ 000 \\ 101 \end{array}, h_7 \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_3 \begin{array}{c} 100 \\ 010 \\ 001 \end{array} =: g \qquad g = eq_3$$

Implementation: $g(x,y) + 2 = f_1(x,y) + f_1(y,x) + h_7(x,y) + u_{\{0,1\}}(x) + u_{\{0,1\}}(y)$

$$\left\{ f_2 := \begin{array}{c} 000 \\ 110 \\ 101 \end{array}, h_7 \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_2 \begin{array}{c} 100 \\ 010 \\ 001 \end{array} =: g \qquad g = eq_3$$

Implementation: $g(x,y) + 1 = f_2(x,y) + h_7(x,y) + u_{\{0\}}(x)$

$$\left\{ f_3 := \begin{array}{c} 000 \\ 010 \\ 001 \end{array} \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_2 \begin{array}{c} 000 \\ 110 \\ 101 \end{array} =: g \qquad g = f_2$$

Implementation: $g(x,y) + 1 = max_z[f_3(z,x) + f_3(z,y) + u_{\{0\}}(y)]$

$$\left\{ f_4 := \begin{array}{c} 001 \\ 110 \\ 001 \end{array} \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_3 \begin{array}{c} 000 \\ 000 \\ 101 \end{array} =: g \qquad g = f_1$$

Implementation: $g(x,y) + 2 = max_z[f_4(z,x) + f_4(y,z) + u_{\{2\}}(z)]$

$$\left\{ f_5 := \begin{array}{c} 000 \\ 010 \\ 101 \end{array} \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_2 \begin{array}{c} 000 \\ 110 \\ 101 \end{array} =: g \qquad g = f_2$$

Implementation: $g(x,y) + 1 = max_z[f_5(x,z) + f_5(y,z) + u_{\{0\}}(y)]$

Case 4

$\mathcal{F} = \{f, h\} \cup \mathcal{U}_D$ and $f \in \mathcal{Q}_1$ and $h \in \mathcal{Q}_0$.

We show that $\{f\} \cup \mathcal{U}_D$ can strictly implement h_7, whereby the result follows from Case 3.

$$\left\{ f_1 := \begin{matrix} 100 \\ 000 \\ 011 \end{matrix} \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_2 \begin{matrix} 100 \\ 011 \\ 011 \end{matrix} =: g \qquad g = h_7$$

Implementation: $g(x,y) + 1 = max_z[f_1(z,x) + f_1(z,y)]$

$$\left\{ f_2 := \begin{matrix} 100 \\ 100 \\ 011 \end{matrix} \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_2 \begin{matrix} 100 \\ 011 \\ 011 \end{matrix} =: g \qquad g = h_7$$

Implementation: $g(x,y) + 1 = max_z[f_2(z,x) + f_2(z,y)]$

$$\left\{ f_3 := \begin{matrix} 100 \\ 100 \\ 000 \end{matrix} \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_3 \begin{matrix} 100 \\ 000 \\ 011 \end{matrix} =: g \qquad g = f_1$$

Implementation:
$g(x,y) + 2 = max_z[f_3(z,x) + f_3(z,y) + f_3(x,y) + u_{\{2\}}(z) + u_{\{2\}}(x) + u_{\{1,2\}}(y)]$

$$\left\{ f_4 := \begin{matrix} 100 \\ 100 \\ 001 \end{matrix} \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_4 \begin{matrix} 100 \\ 100 \\ 000 \end{matrix} =: g \qquad g = f_3$$

Implementation:
$g(x,y) + 3 = max_z[f_4(z,y) + f_4(x,z) + u_{\{2\}}(z) + u_{\{0,1\}}(x) + u_{\{0,1\}}(y)]$

$$\left\{ f_5 := \begin{matrix} 100 \\ 101 \\ 001 \end{matrix} \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_4 \begin{matrix} 100 \\ 100 \\ 001 \end{matrix} =: g \qquad g = f_4$$

Implementation:
$g(x,y) + 3 = max_z[f_5(z,x) + f_5(z,y) + f_5(x,z) + u_{\{0\}}(z) + u_{\{1,2\}}(x)]$

$$\left\{ f_6 := \begin{matrix} 000 \\ 000 \\ 011 \end{matrix} \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_2 \begin{matrix} 100 \\ 100 \\ 000 \end{matrix} =: g \qquad g = f_3$$

Implementation: $g(x,y) + 1 = f_6(x,y) + u_{\{0,1\}}(x) + u_{\{0\}}(y)$

$$\left\{ f_7 := \begin{matrix} 100 \\ 000 \\ 001 \end{matrix} \right\} \cup \mathcal{U}_D \stackrel{s}{\Longrightarrow}_2 \begin{matrix} 100 \\ 101 \\ 001 \end{matrix} =: g \qquad g = f_5$$

Implementation: $g(x,y) + 1 = max_z[f_7(z,x) + f_7(z,y) + u_{\{1\}}(x)]$

Case 5

$\mathcal{F} = \{f, u_{\{0,1\}}\}$ where f is such that $f(0,0) = f(1,1) = 0$ and $f(2,2) = f(0,1) = 1$.

$$\left\{ f_1 := \begin{matrix} 010 \\ 001 \\ 101 \end{matrix}, u_{\{0,1\}} \right\} \stackrel{s}{\Longrightarrow}_3 \begin{matrix} 010 \\ 100 \\ 000 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 01 \\ 10 \end{matrix} \ ([0,1,2] \mapsto [1,0,0])$$

Implementation: $g(x,y)+2 = f_1(x,y)+f_1(y,x)+u_{\{0,1\}}(x)+u_{\{0,1\}}(y)$

$$\left\{ f_2 := \begin{matrix} 011 \\ 000 \\ 011 \end{matrix}, u_{\{0,1\}} \right\} \overset{s}{\Longrightarrow}_3 \begin{matrix} 010 \\ 100 \\ 000 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 01 \\ 10 \end{matrix} \ ([0,1,2] \mapsto [1,0,0])$$

Implementation: $g(x,y)+2 = f_2(x,y)+f_2(y,x)+u_{\{0,1\}}(x)+u_{\{0,1\}}(y)$

$$f_3 := \begin{matrix} 010 \\ 101 \\ 101 \end{matrix} \overset{s}{\Longrightarrow}_3 \begin{matrix} 010 \\ 001 \\ 101 \end{matrix} =: g \qquad g = f_1$$

Implementation: $g(x,y)+2 = max_z[f_3(z,x)+f_3(x,y)+f_3(y,z)]$

$$\left\{ f_4 := \begin{matrix} 010 \\ 000 \\ 001 \end{matrix}, u_{\{0,1\}} \right\} \overset{s}{\Longrightarrow}_4 \begin{matrix} 010 \\ 000 \\ 000 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 01 \\ 00 \end{matrix} \ ([0,1,2] \mapsto [0,1,0])$$

Implementation:
$g(x,y)+3 = max_{z,w}[f_4(z,w)+f_4(w,y)+f_4(x,z)+u_{\{0,1\}}(z)+u_{\{0,1\}}(w)]$

$$\left\{ f_5 := \begin{matrix} 011 \\ 000 \\ 001 \end{matrix}, u_{\{0,1\}} \right\} \overset{s}{\Longrightarrow}_3 \begin{matrix} 011 \\ 000 \\ 011 \end{matrix} =: g \qquad g = f_2$$

Implementation: $g(x,y)+2 = max_{z,w}[f_5(z,y)+f_5(w,z)+f_5(x,w)+u_{\{0,1\}}(z)]$

$$\left\{ f_6 := \begin{matrix} 010 \\ 100 \\ 001 \end{matrix}, u_{\{0,1\}} \right\} \overset{s}{\Longrightarrow}_4 \begin{matrix} 010 \\ 100 \\ 000 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 01 \\ 10 \end{matrix} \ ([0,1,2] \mapsto [1,0,0])$$

Implementation: $g(x,y)+3 = max_{z,w}[f_6(z,w)+f_6(z,y)+f_6(w,x)+u_{\{0,1\}}(z)]$

$$f_7 := \begin{matrix} 011 \\ 100 \\ 001 \end{matrix} \overset{s}{\Longrightarrow}_3 \begin{matrix} 010 \\ 101 \\ 101 \end{matrix} =: g \qquad g = f_3$$

Implementation: $g(x,y)+2 = max_{z,w}[f_7(w,z)+f_7(w,x)+f_7(y,z)]$

$$\left\{ f_8 := \begin{matrix} 010 \\ 001 \\ 001 \end{matrix}, u_{\{0,1\}} \right\} \overset{s}{\Longrightarrow}_4 \begin{matrix} 000 \\ 100 \\ 100 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 00 \\ 10 \end{matrix} \ ([0,1,2] \mapsto [0,1,1])$$

Implementation: $g(x,y)+3 = max_{z,w}[f_8(z,w)+f_8(x,w)+f_8(y,z)+u_{\{0,1\}}(z)]$

$$f_9 := \begin{matrix} 010 \\ 101 \\ 001 \end{matrix} \overset{s}{\Longrightarrow}_3 \begin{matrix} 010 \\ 101 \\ 101 \end{matrix} =: g \qquad g = f_3$$

Implementation: $g(x,y)+2 = max_{z,w}[f_9(w,z)+f_9(w,y)+f_9(x,z)]$

$$\left\{ f_{10} := \begin{matrix} 010 \\ 000 \\ 101 \end{matrix}, u_{\{0,1\}} \right\} \overset{s}{\Longrightarrow}_4 \begin{matrix} 010 \\ 000 \\ 010 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 01 \\ 00 \end{matrix} \ ([0,1,2] \mapsto [0,1,0])$$

Implementation: $g(x,y)+3 = max_{z,w}[f_{10}(z,y)+f_{10}(w,z)+f_{10}(w,x)+u_{\{0,1\}}(z)]$

$$\left\{ f_{11} := \begin{matrix} 011 \\ 000 \\ 101 \end{matrix} , u_{\{0,1\}} \right\} \overset{s}{\Longrightarrow}_4 \begin{matrix} 011 \\ 000 \\ 011 \end{matrix} =: g \qquad g = f_2$$

Implementation: $g(x,y)+3 = max_{z,w}[f_{11}(z,w)+f_{11}(z,y)+f_{11}(w,x)+u_{\{0,1\}}(z)]$

$$\left\{ f_{12} := \begin{matrix} 011 \\ 100 \\ 101 \end{matrix} , u_{\{0,1\}} \right\} \overset{s}{\Longrightarrow}_5 \begin{matrix} 011 \\ 100 \\ 100 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 01 \\ 10 \end{matrix} \; ([0,1,2] \mapsto [1,0,0])$$

Implementation:
$g(x,y)+4 = max_{z,w}[f_{12}(z,w)+f_{12}(z,y)+f_{12}(w,x)+u_{\{0,1\}}(z)+u_{\{0,1\}}(w)]$

$$\left\{ f_{13} := \begin{matrix} 010 \\ 000 \\ 011 \end{matrix} , u_{\{0,1\}} \right\} \overset{s}{\Longrightarrow}_3 \begin{matrix} 011 \\ 000 \\ 011 \end{matrix} =: g \qquad g = f_2$$

Implementation: $g(x,y)+2 = max_{z,w}[f_{13}(z,w)+f_{13}(w,y)+f_{13}(x,z)+u_{\{0,1\}}(z)]$

$$\left\{ f_{14} := \begin{matrix} 010 \\ 001 \\ 011 \end{matrix} , u_{\{0,1\}} \right\} \overset{s}{\Longrightarrow}_4 \begin{matrix} 011 \\ 000 \\ 011 \end{matrix} =: g \qquad g = f_2$$

Implementation: $g(x,y)+3 = max_{z,w}[f_{14}(z,w)+f_{14}(x,z)+f_{14}(y,w)+u_{\{0,1\}}(z)]$

$$\left\{ f_{15} := \begin{matrix} 010 \\ 101 \\ 011 \end{matrix} , u_{\{0,1\}} \right\} \overset{s}{\Longrightarrow}_5 \begin{matrix} 010 \\ 101 \\ 010 \end{matrix} =: g \qquad g \text{ has core } \begin{matrix} 01 \\ 10 \end{matrix} \; ([0,1,2] \mapsto [0,1,0])$$

Implementation:
$g(x,y)+4 = max_{z,w}[f_{15}(z,w)+f_{15}(z,y)+f_{15}(w,x)+u_{\{0,1\}}(z)+u_{\{0,1\}}(w)]$