Introduction to the Maximum Solution Problem

Peter Jonsson[1] and Gustav Nordh[2]

[1] Department of Computer and Information Science, Linköpings Universitet
S-581 83 Linköping, Sweden
petej@ida.liu.se
[2] Laboratoire d'Informatique, École Polytechnique
F-91128 Palaiseau, France
nordh@lix.polytechnique.fr

Abstract. This paper surveys complexity and approximability results for the Maximum Solution (Max Sol) problem. Max Sol is an optimisation variant of the constraint satisfaction problem. Many important and well-known combinatorial optimisation problems are instances of Max Sol: for example, Max Sol restricted to the domain $\{0, 1\}$ is exactly the Max Ones problem (which, in turn, captures problems such as Independent Set and 0/1 Integer Programming). By using this relationship, many different problems in logic, graph theory, integer programming, and algebra can be given a uniform treatment. This opens up for new ways of analysing and solving combinatorial optimisation problems.

1 Introduction

A large number of natural optimisation problems can be viewed as instances of the Maximum Solution (Max Sol) problem. In this survey, we will describe the problem and present complexity and approximability results. This section provides the formal definition together with some background information. Some additional material on approximability and universal algebra can be found in Section 2. All results for Max Sol on Boolean domains are collected in Section 3. In order to familiarise the reader with the basic tools, Section 3 also contains a derivation of new, sharper inapproximability bounds for the Boolean Max Sol problem. Sections 4–9 contain results for the Max Sol problem on non-Boolean domains; the examples come from many different areas such as logic, graph theory, and algebra. To make the survey more readable, we devote the first of these sections to two general tractability results. The Max Sol problem is compared to other optimisation formalisms in Section 10, and some open questions are posed in Section 11.

1.1 Background and basic assumptions

We introduce Max Sol by first considering the well-known Max Ones problem [35]: an instance of Max Ones consists of constraints applied to a number

of Boolean variables, and the goal is to find an assignment that satisfies all constraints while maximising the number of variables set to 1. The only difference between MAX ONES and MAX SOL is that we do not require the domain of the variables to be Boolean—the domain of the variables in MAX SOL are allowed to be any subset of the natural numbers, and the objective is to find an assignment that satisfy all the constraints and that maximise the sum of the variables. We parameterise the problem according to the set of allowed constraint types, i.e. for any set Γ of relations, MAX SOL(Γ) denotes the set of problems where the constraint types are restricted to Γ. The main goal of this survey is to present complexity and approximability results for MAX SOL(Γ) under different choices of Γ.

Let us now take a look at the maximisation problem INTEGER PROGRAMMING:

Instance: $m \times n$ matrix A of rationals, m-vector b of rationals, and n-vector c of rationals.
Solution: An n-vector x of integers such that $Ax \geq b$ and $x \geq 0$;
Measure: $c^T x$.

Obviously, one can view the integer programming problem as a MAX SOL(Γ) problem for a certain constraint language Γ over the integers. However, we will restrict ourselves to a certain subclass of MAX SOL problems in this survey; in fact, this class do *not* contain INTEGER PROGRAMMING. The restrictions we encompass are two finiteness conditions: we only consider finite domains and finite constraint languages. This enables us to use algebraic techniques for studying MAX SOL.

The finiteness conditions do not prevent us from capturing interesting problems, though. It is easy to see that MAX SOL over the domain $\{0, 1\}$ captures, for instance, MAX INDEPENDENT SET (problem GT23 in [2]), and certain variants of MAX 0/1 PROGRAMMING (problem MP2 in [2]). There are also many interesting non-Boolean MAX SOL problems: examples include problems in integer programming [25], multiple-valued logic [33], and equation solving over Abelian groups [36].

Only considering finite constraint languages may seem quite restrictive but, in many cases, it is not. Consider for instance integer programming over the bounded domain $\{0, \ldots, d-1\}$, i.e., the size of the domain is bounded by a constant but the length of the inequalities are not. Each row in the constraint matrix can be viewed as an inequality

$$a_1 x_1 + a_2 x_2 + \ldots + a_k x_k \geq b.$$

Obviously, such an inequality is equivalent to the following three inequalities

$$\begin{gathered} a_1 x_1 + a_2 x_2 + \ldots + a_{\lfloor k/2 \rfloor} x_{\lfloor k/2 \rfloor} - z \geq 0 \\ -a_1 x_1 - a_2 x_2 - \ldots - a_{\lfloor k/2 \rfloor} x_{\lfloor k/2 \rfloor} + z \geq 0 \\ z + a_{\lfloor k/2 \rfloor + 1} + \ldots + a_k x_k \geq b \end{gathered}$$

where z denotes a fresh variable that is given the weight 0 in the objective function. By repeating this process, one ends up with a set of inequalities where each inequality contains at most three variables, and the optimal solution to this instance have the same measure as the original instance. There are at most $2^d + 2^{d^2} + 2^{d^3}$ different relations (and thus inequalities) of length ≤ 3 on a d element domain. Since the size of the domain is constant, we have reduced the problem to one with a finite constraint language. Finally, this reduction is polynomial-time: each inequality of length k in the original instance give rise to at most $3^{\lceil \log_2 k \rceil} = O(k^2)$ inequalities and at most $O(k^2)$ new variables.

The restriction to finite domains appears to be more problematic since it provably excludes certain prominent problems (such as unbounded Integer Programming). There has been some efforts lately in order to make the algebraic framework applicable to infinite-domain problems [4]. To the best of our knowledge, such extended methods have not been applied to the Max Sol problem.

1.2 Formal definition

Let us now formally define Max Sol: let $D \subset \mathbb{N}$ (*the domain*) be a finite set. The set of all n-tuples of elements from D is denoted by D^n. Any subset of D^n is called an n-ary relation on D. The set of all finitary relations over D is denoted by R_D. A *constraint language* over a finite set, D, is a finite set $\Gamma \subseteq R_D$. Constraint languages are the way in which we specify restrictions on our problems. The constraint satisfaction problem over the constraint language Γ, denoted Csp(Γ), is defined to be the decision problem with instance (V, D, C), where

- V is a set of variables,
- D is a fixed finite set of values (sometimes called a domain), and
- C is a set of constraints $\{C_1, \ldots, C_q\}$, in which each constraint C_i is a pair (s_i, R_i) where s_i is a list of variables of length m_i, called the constraint scope, and R_i is an m_i-ary relation over the set D, belonging to Γ, called the constraint relation.

The question is whether there exists a solution to (V, D, C) or not, that is, a function from V to D such that, for each constraint in C, the image of the constraint scope is a member of the constraint relation. To exemplify this definition, let NAE be the following ternary relation on $\{0, 1\}$: $NAE = \{0, 1\}^3 \setminus \{(0, 0, 0), (1, 1, 1)\}$. It is easy to see that the well-known **NP**-complete problem Not-All-Equal 3-Sat can be expressed as Csp($\{NAE\}$).

The optimisation problem that we are going to study, Weighted Maximum Solution(Γ) (which we abbreviate Max Sol(Γ)) is defined as follows:

Instance: Tuple (V, D, C, w), where D is a fixed finite subset of $\mathbb{N}$, (V, D, C) is a Csp(Γ) instance, and $w : V \to \mathbb{N}$ is a weight function.
Solution: An assignment $f : V \to D$ to the variables such that all constraints are satisfied.

Measure: $\sum_{v \in V} w(v) \cdot f(v)$

Note that in the definition of the MAX SOL(Γ)) problem, the domain D is part of the input even though it is implicitly defined by the constraint language Γ. From a complexity point of view, this does not matter since the domain is defined to be a fixed finite set and the size of the domain is constant.

We illustrate this definition with a simple example:

Example 1. Consider the domain $D = \{0, 1\}$ and the binary relation $R = \{(0, 0), (1, 0), (0, 1)\}$. Then, MAX SOL($\{R\}$) is exactly the weighted MAX INDEPENDENT SET problem.

Several related problems can be defined along the same lines, e.g. MIN SOL where the objective is to minimise $\sum_{v \in V} w(v) \cdot f(v)$ and MAX AW SOL where we allow the weight function w to be a function from V to the integers $\mathbb{Z}$. The Boolean variants of these problems have been studied earlier [30, 35]. Note that for Boolean constraint languages Γ, the MAX SOL(Γ) problem is usually denoted MAX ONES(Γ).

There are several aspects of the definition of MAX SOL that can be discussed. Below, we consider two points that have been questioned and/or criticised in the past.

I. Choice of measure function. Note that our choice of measure function in the definition of MAX SOL(Γ) is just one of several reasonable choices. Another reasonable alternative, used in [36], would be to let the domain D be any finite set and introduce an additional function $g : D \to \mathbb{N}$ mapping elements from the domain to natural numbers. The measure could then be defined as $\sum_{v \in V} w(v) \cdot g(f(v))$. This would result in a parameterised problem MAX SOL(Γ, g) where the goal is to classify the complexity of MAX SOL(Γ, g) for all combinations of constraint languages Γ and functions g. Note that our definition of MAX SOL(Γ) is equivalent to the definition of MAX SOL(Γ, g) if in addition g is required to be injective. Our main motivation for the choice of measure function is to stay close to integer programming and MAX ONES. However, we will use the alternative definition when studying equations over Abelian groups in Section 7.

II. Weighted vs. unweighted problems. We remark that we do not deal explicitly with the unweighted version of the problem (in the unweighted version, all variables have weight 1). The correspondence, in terms of approximability, between the weighted and unweighted versions of the problem has already been discussed in depth [13, 35]. In summary, Khanna *et al.* [35] prove that if MAX SOL(Γ) is in **poly-APX**, then hardness results (for the weighted version) implies the corresponding hardness results for the unweighted version. The basic idea of the proof is to simulate weights by replication of variables.

Moreover, since all tractability results for MAX SOL(Γ) (that we are aware of) hold for the weighted version of the problem, it should be clear that the classifications we report here also hold for the unweighted version of the MAX

Sol(Γ) problem. But, in general, it is still open whether tractability for the unweighted version implies tractability of the weighted version of the Max Sol(Γ) problem. The correspondence between unweighted and weighted problems does not hold if negative weights are allowed [30].

1.3 Methods

The complexity and approximability of Max Ones(Γ) are completely known for all choices of Γ [35]. For any Boolean constraint language Γ, Max Ones(Γ) is either in **PO** or is **APX**-complete or **poly-APX**-complete or finding a solution of non-zero value is **NP**-hard or finding any solution is **NP**-hard. The exact borderlines between the different cases are given in [35]. For larger domains, it seems significantly harder to obtain an exact characterisation of approximability than in the Boolean case. Such a characterisation would, for instance, show whether the dichotomy conjecture for constraint satisfaction problems is true or not – a famous open question which is believed to be difficult [16]. Hence, it seems reasonable to study restricted (but as general as possible) families of constraint languages where the complexity and approximability can be determined. In doing so, the *algebraic approach* appears to be indispensable. When the algebraic approach is applicable to a certain problem, there is an equivalence relation on the constraint languages such that two constraint languages which are equivalent under this relation have the same complexity. More specifically, two constraint languages are in the same equivalence class if they generate the same *relational clone*. The relational clone generated by Γ, captures the expressive power of Γ and is denoted by $\langle\Gamma\rangle$. Hence, instead of studying every possible finite set of relations it is enough to study the relational clones.

The algebraic approach is known to be applicable to Max Sol. In fact, it is known that constraint languages Γ_1 and Γ_2 such that $\langle\Gamma_1\rangle = \langle\Gamma_2\rangle$, then Max Sol($\Gamma_1$) S-reduces to Max Sol(Γ_2), and vice-versa. An *S-reduction* is a certain strong approximation-preserving reduction: if $\langle\Gamma_1\rangle = \langle\Gamma_2\rangle$, then Γ_1 and Γ_2 are very similar with respect to approximability. For instance, if Max Sol(Γ_1) is **NP**-hard to approximate within some constant c, then Max Sol(Γ_2) is **NP**-hard to approximate within c, too. We note that the clone-theoretic approach was not used in the original classification of Max Ones.

2 Preliminaries

The purpose of this section is to provide a brief overview of approximability and the algebraic approach. We refer the reader to [2] for a deeper treatment of approximability and to [7, 41] for a deeper treatment of the algebraic approach.

2.1 Approximability, reductions, and completeness

A *combinatorial optimisation problem* is defined over a set of *instances* (admissible input data); each instance I has a finite set $\mathsf{sol}(I)$ of *feasible solutions*

associated with it. Given an instance I and a feasible solution s of I, $m(I, s)$ denotes the positive integer *measure* of s. The objective is, given an instance I, to find a feasible solution of *optimum* value with respect to the measure m. The optimal value is the largest one for *maximisation* problems and the smallest one for *minimisation* problems. A combinatorial optimisation problem is said to be an **NPO** problem if its instances and solutions can be recognised in polynomial time, the solutions are polynomially bounded in the input size, and the objective function can be computed in polynomial time (see, e.g., [2]).

We say that a solution $s \in \mathsf{sol}(I)$ to an instance I of an **NPO** problem Π is *r-approximate* if it is satisfying

$$\max\left\{\frac{m(I,s)}{\text{OPT}(I)}, \frac{\text{OPT}(I)}{m(I,s)}\right\} \leq r,$$

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

We define **PO** to be 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. Similarly, Π is in the class **poly-APX** if there is a polynomial-time approximation algorithm for Π whose performance ratio is bounded by a polynomial in the size of the input. Completeness in **APX** and **poly-APX** is defined using appropriate reductions, called AP-reductions and A-reductions, respectively [12, 35]. AP-reductions are more sensitive than A-reductions and every AP-reduction is also an A-reduction [35]. In this paper we will not need the added flexibility of A-reductions for proving our **poly-APX**-completeness results. Hence, we only need the definition of AP-reductions.

Definition 2. *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 I of Π_1, $F(I)$ is an instance of Π_2;
(b) for any instance I of Π_1, and any feasible solution s' of $F(I)$, $G(I, s')$ is a feasible solution of I;
(c) for any instance I of Π_1, and any $r \geq 1$, if s' is an r-approximate solution of $F(I)$ then $G(I, s')$ is an $(1 + (r - 1)\alpha + o(1))$-approximate solution of I where the o-notation is with respect to $|I|$.

In some cases we will use another kind of reduction, S-reductions. They are defined as follows:

Definition 3. *An* **NPO** *problem Π_1 is said to be* S-reducible *to an* **NPO** *problem Π_2 if two polynomial-time computable functions F and G exist such that*

(a) given any instance I of Π_1, algorithm F produces an instance $I' = F(I)$ of Π_2, such that the measure of an optimal solution for I', $\text{OPT}(I')$, is exactly $\text{OPT}(I)$.

(b) given $I' = F(I)$, and any solution s' to I', algorithm G produces a solution s to I such that $m_1(I, G(s')) = m_2(I', s')$, where m_1 is the measure for Π_1 and m_2 is the measure for Π_2.

Obviously, the existence of an S-reduction from Π_1 to Π_2 implies the existence of an AP-reduction from Π_1 to Π_2. The reason why we need S-reductions is that AP-reductions do not (generally) preserve membership in **PO** [35]. We also note that S-reduction preserve approximation thresholds exactly for problems in **APX**: let Π_1, Π_2 be problems in **APX**, assume that it is **NP**-hard to approximate Π_1 within c, and that there exists an S-reduction from Π_1 to Π_2. Then, it is **NP**-hard to approximate Π_2 within c, too.

2.2 Algebraic approach

We begin by giving a number of basic definitions. An *operation* on a finite set D (the domain) is an arbitrary function $f : D^k \to D$. Any operation on D can be extended in a standard way to an operation on tuples over D, as follows: let f be a k-ary operation on D and let R be an n-ary relation over D. For any collection of k tuples, $\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k} \in R$, the n-tuple $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k})$ is defined as follows: $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k}) = (f(\boldsymbol{t_1}[1], \boldsymbol{t_2}[1], \ldots, \boldsymbol{t_k}[1]), f(\boldsymbol{t_1}[2], \boldsymbol{t_2}[2], \ldots, \boldsymbol{t_k}[2]), \ldots, f(\boldsymbol{t_1}[n], \boldsymbol{t_2}[n], \ldots, \boldsymbol{t_k}[n]))$, where $\boldsymbol{t_j}[i]$ is the i-th component in tuple $\boldsymbol{t_j}$. A technique that has been shown to be useful in determining the computational complexity of $\textsc{Csp}(\Gamma)$ is that of investigating whether the constraint language Γ is invariant under certain families of operations [28].

Now, let $R_i \in \Gamma$. If f is an operation such that for all $\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k} \in R_i$ $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k}) \in R_i$, then R_i is *invariant* (or, in other words, closed) under f. If all constraint relations in Γ are invariant under f then Γ is invariant under f. An operation f such that Γ is invariant under f is called a *polymorphism* of Γ. The set of all polymorphisms of Γ is denoted $Pol(\Gamma)$. Given a set of operations F, the set of all relations that are invariant under all the operations in F is denoted $Inv(F)$. Whenever there is only one operation under consideration, we write $Inv(f)$ instead of $Inv(\{f\})$.

We will need a number of operations in the sequel: an operation f over D is said to be

- a *constant* operation if f is unary and $f(a) = c$ for all $a \in D$ and some $c \in D$;
- a *majority* operation if f is ternary and $f(a, a, b) = f(a, b, a) = f(b, a, a) = a$ for all $a, b \in D$;
- a *binary commutative idempotent* operation if f is binary, $f(a, a) = a$ for all $a \in D$, and $f(a, b) = f(b, a)$ for all $a, b \in D$;
- an *affine* operation if f is ternary and $f(a, b, c) = a - b + c$ for all $a, b, c \in D$ where $+$ and $-$ are the binary operations of an Abelian group $(D, +, -)$.

Example 4. Let $D = \{0, 1, 2\}$ and let f be the majority operation on D where $f(a, b, c) = a$ if a, b and c are all distinct. Furthermore, let

$$R = \{(0, 0, 1), (1, 0, 0), (2, 1, 1), (2, 0, 1), (1, 0, 1)\}.$$

It is easy to verify that for every triple of tuples, $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in R$, we have $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \in R$. For example, if $\boldsymbol{x} = (0,0,1), \boldsymbol{y} = (2,1,1)$, and $\boldsymbol{z} = (1,0,1)$, then

$$f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \Big(f(\boldsymbol{x}[1], \boldsymbol{y}[1], \boldsymbol{z}[1]), f(\boldsymbol{x}[2], \boldsymbol{y}[2], \boldsymbol{z}[2]), f(\boldsymbol{x}[3], \boldsymbol{y}[3], \boldsymbol{z}[3]) \Big) = \\ \big(f(0,2,1), f(0,1,0), f(1,1,1) \big) = (0,0,1) \in R.$$

We can conclude that R is invariant under f or, equivalently, that f is a polymorphism of R.

We sometimes need to define relations in terms of other relations, using certain logical formulas. In doing so, the algebraic approach provides a convenient tool. A first-order formula ϕ over a constraint language Γ is said to be *primitive positive* (or *pp-formula* for short) if it is of the form

$$\exists \mathbf{x} : (R_1(\mathbf{x}_1) \wedge \ldots \wedge R_k(\mathbf{x}_k))$$

where $R_1, \ldots, R_k \in \Gamma$ and $\mathbf{x}_1, \ldots, \mathbf{x}_k$ are vectors of variables such that the arity of R_i equals the length of the vector $\mathbf{x}_i$ for all i. Note that a pp-formula ϕ with m free variables defines an m-ary relation $R \subseteq D^m$, denoted $R \equiv_{pp} \phi$; the relation R is the set of all m-tuples satisfying the formula ϕ.

We continue by defining a closure operation $\langle \cdot \rangle$ on sets of relations: for any set $\Gamma \subseteq R_D$ the set $\langle \Gamma \rangle$ consists of all relations that can be expressed using relations from $\Gamma \cup \{=_D\}$ ($=_D$ is the equality relation on D), conjunction, and existential quantification, i.e. $\langle \Gamma \rangle$ is the set of all relations that can be expressed via pp-formulas over Γ. The sets of relations of the form $\langle \Gamma \rangle$ are referred to as *relational clones*. There is a very strong connection between a relational clone $\langle \Gamma \rangle$ and the operators that Γ is invariant under:

Theorem 5 ([37]). *For every set $\Gamma \subseteq R_D$, $\langle \Gamma \rangle = Inv(Pol(\Gamma))$.*

The following result shows that the algebraic approach is applicable when studying the approximability of Max Sol(Γ):

Theorem 6 ([33]). *Let Γ be a constraint language and $\Gamma' \subseteq \langle \Gamma \rangle$ finite. Then,* Max Sol(Γ') *is S-reducible to* Max Sol(Γ).

The concept of *cores* has been important when studying the complexity of Csp; a constraint language is a core if every unary polymorphism is injective (i.e. a permutation). We will use a related concept (that was introduced in [33]) when studying Max Sol.

Definition 7. *A constraint language Γ is a* max-core *if and only if there is no non-injective unary operation f in $Pol(\Gamma)$ such that $f(d) \geq d$ for all $d \in D$. A constraint language Γ' is a max-core of Γ if and only if Γ' is a max-core and $\Gamma' = f(\Gamma)$ for some unary operation $f \in Pol(\Gamma)$ such that $f(d) \geq d$ for all $d \in D$.*

The constraint language $\{R\}$ where $R = \{(0,0),(1,1),(2,1),(1,2)\}$ is *not* a max-core since it admits a unary polymorphism $f : \{0,1,2\} \to \{0,1,2\}$ defined such that $f(0) = 1$, $f(1) = 1$, and $f(2) = 2$. It is easy to see that $\{R'\}$ is a max-core of $\{R\}$ where $R' = \{(1,1),(2,1),(1,2)\}$.

Lemma 8 ([33]). *If Γ' is a max-core of Γ, then* Max Sol(Γ) *and* Max Sol(Γ') *are polynomial-time equivalent.*

3 Boolean domain

The Max Sol(Γ) problem over Boolean constraint languages Γ goes under the name Max Ones(Γ) and the approximability for every finite Boolean constraint language has been classified by Khanna *et al.* in [35]. Before stating this result, we recall the following standard restrictions on Boolean constraint languages (see, e.g. [12]).

Definition 9.

- *Γ is* 0*-valid if* $(0,0,\ldots,0) \in R$ *for every relation R in Γ,*
- *Γ is* 1*-valid if* $(1,1,\ldots,1) \in R$ *for every relation R in Γ,*
- *Γ is* Horn *if every relation R in Γ is the set of models of a CNF formula having at most one unnegated variable in each clause,*
- *Γ is* dual Horn *if every relation R in Γ is the set of models of a CNF formula having at most one negated variable in each clause,*
- *Γ is* bijunctive *if every relation R in Γ is the set of models of a CNF formula having at most two literals in each clause,*
- *Γ is* affine *if every relation R in Γ is the set of models of a system of linear equations over $GF(2)$, the field with two elements,*
- *Γ is* width-2 affine *if every relation R in Γ is the set of models of a system of linear equations over $GF(2)$ in which each equation has at most 2 variables.*

Theorem 10 ([35]). *Given a constraint language Γ over the Boolean domain $\{0,1\}$,*

- *if Γ is* 1*-valid or dual-Horn or width-*2 *affine, then* Max Sol(Γ) *is in* **PO***;*
- *otherwise if Γ is affine, then* Max Sol(Γ) *is* **APX***-complete;*
- *otherwise if Γ is Horn or bijunctive, then* Max Sol(Γ) *is* **poly-APX***-complete;*
- *otherwise if Γ is* 0*-valid, then finding a solution of positive measure is* **NP***-hard;*
- *otherwise finding a feasible solution to* Max Sol(Γ) *is* **NP***-hard.*

We remark that Khanna *et al.*'s proof of the preceding theorem does not make use of the algebraic approach via relational clones. The bulk of their proof is spent on showing numerous explicit implementations/reductions between different constraint languages. The advantage of using the algebraic approach is that there is no need to deal with these implementations/reductions explicitly.

In fact, it is quite easy to strengthen Khanna *et al.*'s result and give tight approximability thresholds for Max Sol(Γ) over Boolean constraint languages.

Theorem 11. *Given a constraint language Γ over $\{0,1\}$,*

- *if Γ is 1-valid or dual-Horn or width-2 affine, then* Max Sol(Γ) *is in* **PO**;
- *otherwise if Γ is affine, then* Max Sol(Γ) *is 2-approximable but not approximable to within $2-\varepsilon$ for any $\varepsilon > 0$ unless* **P**=**NP**;
- *otherwise if Γ is Horn or bijunctive, then* Max Sol(Γ) *is approximable to within $O(|V|)$ but not to within $O(|V|^{1-\varepsilon})$ for any $\varepsilon > 0$ unless* **P** = **NP**;
- *otherwise if Γ is 0-valid, then finding a solution of positive measure is* **NP***-hard;*
- *otherwise finding a feasible solution to* Max Sol(Γ) *is* **NP***-hard.*

All the bounds except the $2-\varepsilon$ hardness for affine constraint languages (that are not 1-valid) and the $O(|V|^{1-\varepsilon})$-hardness for Horn and bijunctive constraint languages are proved in (or easily follows from) [35]. The $2-\varepsilon$ hardness for affine constraint languages consisting of relations expressed by equations on the form

$$\{x_1 + \cdots + x_k = c \pmod 2 \mid k \text{ even}, c \in \{0,1\}\}$$

follows from a more general result due to Kuivinen [36]. The proof in [36] consists of a gap preserving reduction from the problem Max-E3-Lin-2. Max-E3-Lin-2 is the following problem: given a set of equations over $\mathbb{Z}_2$ with exactly three variables per equation, satisfy as many equations as possible. It is proved in [26] that it is **NP**-hard to approximate Max-E3-Lin-2 within $2-\varepsilon$ for any $\varepsilon > 0$.

The remaining affine constraint languages that need to be classified consist of relations expressed by equations on the form

$$\{x_1 + \cdots + x_k = 0 \pmod 2 \mid k \in \mathbb{N}\}.$$

The $2-\varepsilon$ hardness for these constraint languages can be proved by modifying the corresponding reduction in [36]. For more details, please consult [32].

The $O(|V|^{1-\varepsilon})$-hardness for Horn and bijunctive constraint languages (that are neither 1-valid, nor dual-Horn, nor affine) may seem trivial at first sight since Theorem 6 together with the inclusion structure among Boolean relational clones tells us that there is an S-reduction from Max Independent Set to Max Sol over any such constraint language. But please keep in mind that S-reductions do not (in general) preserve instance size. For example, if Max Sol(Γ_1) is **NP**-hard to approximate within $O(|V|)$ and there is an S-reduction from Max Sol(Γ_1) to Max Sol(Γ_2) which blows up the instance size by a quadratic factor. Then, this only implies that Max Sol(Γ_2) is **NP**-hard to approximate within $O(|V|^2)$. Hence, we cannot use the S-reduction implied by Theorem 6 directly since it may increase instance sizes by a polynomial factor. Fortunately, it is possible with some extra work to give an S-reduction from Max Independent Set to Max Sol(Γ), which increase instance sizes by at most a factor 2, for Horn and bijunctive constraint languages (that are neither 1-valid, nor dual-Horn, nor affine). In fact, in the Horn case it is possible to give an S-reduction which only introduces two extra variables. Hence, the $O(|V|^{1-\varepsilon})$-hardness result follows from the corresponding hardness result for Max Independent Set [26, 44].

4 Tractability results

In this section, we present tractability results for two classes of constraint languages: *injective* constraint languages and *generalised max-closed* constraint languages. These two classes includes almost all of the known tractable classes of constraint languages presented in the literature for this problem. In particular, they can be seen as substantial and nontrivial generalisations of the tractable classes known for the corresponding Max Ones problem over the Boolean domain. We have already (in Section 3) pointed out that there are only three tractable classes of constraint languages over the Boolean domain: width-2 affine, 1-valid, and dual-Horn [35]. Width-2 affine constraint languages are examples of injective constraint languages and the classes of 1-valid and dual-Horn constraint languages are examples of generalised max-closed constraint languages. The monotone constraints which are, for instance, studied by Hochbaum *et al.* [24, 25] (in relation with integer programming) and Woeginger [42] (in relation with constraint satisfaction) are also related to generalised max-closed constraints. Hochbaum & Naor [25] show that monotone constraints can be characterised as those constraints that are simultaneously invariant under the max and min operators. Hence, monotone constraints are also generalised max-closed constraints as long as the underlying domain is finite.

4.1 Injective relations

We begin by formally defining *injective relations*.

Definition 12. *A binary relation, $R \in R_D$, is called* injective *if there exists a subset $D' \subseteq D$ and an injective function $\pi : D' \to D$ such that*

$$R = \{(x, \pi(x)) \mid x \in D'\}.$$

It is important to note that the function π is *not* assumed to be total on D. Let I^D denote the set of all injective relations on the domain D and let $\Gamma_I^D = \langle I^D \rangle$. We say that a constraint language Γ is injective if $\Gamma \subseteq \Gamma_I^D$.

Example 13. Let $D = \{0, 1\}$ and let $R = \{(x, y) \mid x, y \in D,\ x + y \equiv 1 \pmod 2\}$. R is injective because the function $f : D \to D$ defined as $f(0) = 1$ and $f(1) = 0$ is injective. More generally, let $G = (D', +, -)$ be an arbitrary Abelian group and let $c \in D'$ be an arbitrary group element. It is easy to see that the relation $\{(x, y) \mid x, y \in D',\ x + y = c\}$ is injective.

R is an example of a relation which is invariant under an affine operation. Such relations have previously been studied in relation with the Max Ones problem in [36]. We will give some additional results for such constraints in Section 7. With the terminology used in [36], R is said to be *width-2 affine*. The relations which can be expressed as the set of solutions to an equation with two variables over an Abelian group are exactly the width-2 affine relations in [36], so the injective relations are a superset of the width-2 affine relations. The following result is a direct consequence of [10, Sec. 4.4].

Theorem 14. *If Γ is injective, then* Max Sol(Γ) *is in* **PO**.

An alternative way of proving Theorem 14 goes like this: Max Sol(Γ_I^D) is in **PO** if and only if Max Sol(I^D) is in **PO** (by Theorem 6) so we can concentrate on I^D. Given an instance of Max Sol(I^D), consider the graph having the variables as vertices and edges between the vertices/variables occurring together in the same constraint. Each connected component of this graph represents an independent subproblem that can be solved in separately. If a value is assigned to a variable/vertex, all variables/vertices in the same component will be forced to take a value by propagating this assignment. Hence, each connected component have at most $|D|$ different solutions (that can be easily enumerated) and an optimal one can be found in polynomial time.

Injective relations can also be defined via a polymorphism: define the *discriminator* $t : D^3 \to D$ such that

$$t(a,b,c) = \begin{cases} c & \text{if } a = b, \\ a & \text{otherwise.} \end{cases}$$

It is known that $\Gamma_I^D = Inv(t)$ [41, Theorem 4.2].

4.2 Generalised max-closed relations

We begin by giving the basic definition.

Definition 15. *A constraint language Γ over a domain $D \subset \mathbb{N}$ is* generalised max-closed *if and only if there exists a binary operation $f \in Pol(\Gamma)$ such that f satisfies the following two conditions:*

1. *for all $a, b \in D$ such that $a \neq b$ it holds that if $f(a,b) \leq \min(a,b)$, then $f(b,a) > \max(a,b)$; and*
2. *for all $a \in D$ it holds that $f(a,a) \geq a$.*

The following two examples will clarify the definition above.

Example 16. Assume that the domain D is $\{0,1,2,3\}$. As an example of a generalised max-closed relation consider $R = \{(0,0),(1,0),(0,2),(1,2)\}$. R is invariant under max and is therefore generalised max-closed as max satisfies the properties of Definition 15. Now, consider the relation Q defined as

$$Q = \{(0,1),(1,0),(2,1),(2,2),(2,3)\}.$$

Q is not invariant under max because

$$\max((0,1),(1,0)) = (\max(0,1),\max(1,0)) = (1,1) \notin Q.$$

Let the operation $\circ : D^2 \to D$ be defined by the following Cayley table (note that we write $x \circ y$ instead of $\circ(x,y)$):

$\circ$	0	1	2	3
0	0	2	2	3
1	2	1	2	2
2	2	2	2	3
3	3	2	3	3

Now, it is easy to verify that $Inv(\circ)$ is a set of generalised max-closed relations and that $Q \in Inv(\circ)$.

Example 17. Consider the relations R_1 and R_2 defined as,

$$R_1 = \{(1,1,1),(1,0,0),(0,0,1),(1,0,1)\}$$

and $R_2 = R_1 \setminus \{(1,1,1)\}$. The relation R_1 is 1-valid because the tuple consisting only of ones is in R_1, i.e., $(1,1,1) \in R_1$. The relation R_2, on the other hand, is not 1-valid but is dual-Horn positive because it is invariant under max. Note that both R_1 and R_2 are generalised max-closed since R_1 is invariant under $f(x,y) = 1$ and R_2 is invariant under $f(x,y) = \max(x,y)$. It is in fact the case that every dual-Horn relation is invariant under max so the 1-valid and dual-Horn relations are subsets of the generalised max-closed relations.

We are now ready to explain the tractability of generalised max-closed constraint languages.

Theorem 18 ([32]). *If Γ is generalised max-closed, then* Max Sol(Γ) *is in* **PO**.

The tractability of generalised max-closed constraint languages crucially depends on the following property. If Γ is generalised max-closed, then all relations

$$R = \{(d_{11}, d_{12}, \ldots, d_{1m}), \ldots, (d_{t1}, d_{t2}, \ldots, d_{tm})\}$$

in Γ have the property that the tuple

$$\boldsymbol{t}_{\max} = (\max\{d_{11}, \ldots, d_{t1}\}, \ldots, \max\{d_{1m}, \ldots, d_{tm}\})$$

is in R, too.

This property is the basis for the simple consistency based algorithm for Csps over max-closed constraint languages, from [29]. We use the same algorithms to solve Max Sol(Γ) when Γ is generalised max-closed. The algorithm for Csps over max-closed constraint languages from [29], gives us a solution (if one exists) which has the property that the value assigned to each variable is the maximum value this variable is allowed to take in any solution. Hence, this solution is also the optimum solution to the corresponding Max Sol problem. Since the only property of max-closed constraint languages that is exploited by this algorithm is that the tuple $\boldsymbol{t}_{\max}$ is in every relation, it follows that the same algorithm solves (to optimum) Max Sol(Γ) for generalised max-closed constraint languages.

5 Clausal constraints

We will now introduce our first example of a non-Boolean Max Sol problem. We consider a framework for expressing constraint languages based on regular signed logic [23] over totally-ordered sets. This approach was introduced by Creignou *et al.* [11]. The set of relations that we consider is based on *regular*

signed logic [23], where the underlying domain is a (possibly infinite) totally-ordered set of integers $\{0, 1, \ldots\}$. This logic provides us with convenient concepts for defining a class of relations with strong modelling capabilities. Jeavons and Cooper [29] have proved that any constraint can be expressed as the conjunction of expressions over this class of relations. A disadvantage with their approach is that the resulting set of constraints may by exponentially large (in the number of tuples in the constraint to be expressed). An improved algorithm solving the same problem has been suggested by Gil *et al.* [17]. It takes a constraint/relation represented by the set of all assignments/tuples that satisfies it and outputs in polynomial time (in the number of tuples) an expression that is equivalent to the original constraint.

Let V be a set of variables. For $x \in V$ and $a \in D$, the inequalities $x \geq a$ and $x \leq a$ are called positive and negative literals, respectively. A *clause* is a disjunction of literals. A *clausal pattern* is a multiset of the form $P = (+a_1, \ldots, +a_p, -b_1, \ldots, -b_q)$ where $p, q \in \mathbb{N}$ and $a_i, b_i \in D$ for all i. The pattern P is said to be *negative* if $p = 0$ and *positive* if $q = 0$. The sum $p+q$, also denoted $|P|$, is the *length* of the pattern.

A *clausal language* L is a set of clausal patterns. Given a clausal language L, an *L-clause* is a pair $(P, \mathbf{x})$, where $P \in L$ is a pattern and $\mathbf{x}$ is a vector of not necessarily distinct variables from V such that $|P| = |\mathbf{x}|$. A pair $(P, \mathbf{x})$ with a pattern $P = (+a_1, \ldots, +a_p, -b_1, \ldots, -b_q)$ and variables $\mathbf{x} = (x_1, \ldots, x_{p+q})$ represents the clause

$$(x_1 \geq a_1 \vee \ldots \vee x_p \geq a_p \vee x_{p+1} \leq b_1 \vee \ldots \vee x_{p+q} \leq b_q),$$

where $\vee$ is the disjunction operator. An L-formula ϕ is a conjunction of a finite number of L-clauses. An *assignment* is a mapping $I : V \to D$ assigning a domain element $I(x)$ to each variable $x \in V$. An assignment I *satisfies* an L-formula ϕ if and only if

$$(I(x_1) \geq a_1 \vee \ldots \vee I(x_p) \geq a_p \vee I(x_{p+1}) \leq b_1 \vee \ldots \vee I(x_{p+q}) \leq b_q)$$

holds for every clause in ϕ. It can be easily seen that the literals $+0$ and $-d$ are superfluous since the inequalities $x \geq 0$ and $x \leq d$ vacuously hold. Without loss of generality, it is sufficient to only consider patterns and clausal languages without such literals. We see that clausal patterns are nothing more than a convenient way of specifying certain relations — consequently, we can use them for defining constraint languages. Thus, we make the following definitions: given a clausal language L and a clausal pattern $P = (+a_1, \ldots, +a_p, -b_1, \ldots, -b_q)$, we let $Rel(P)$ denote the corresponding relation, i.e. $Rel(P) = \{\mathbf{x} \in D^{p+q} \mid (P, \mathbf{x}) \text{ hold}\}$ and $\Gamma_L = \{Rel(P) \mid P \in L\}$.

It is easy to see that several well-studied optimisation problems are captured by this framework.

Example 19. Let the domain D be $\{0, 1\}$. The problem Independent Set (where the objective is to find an independent set of maximum weight in an undirected graph) can be viewed as the Max Sol($\Gamma_{(-0,-0)}$) problem. Similarly,

Max Sol($\Gamma_{(-0,\ldots,-0)}$) (with k literals) is the Max k-Hypergraph Stable Set problem.

We can now present sufficient conditions for when Max Sol over clausal languages is tractable and prove that it is **APX**-hard otherwise. To do so, we use a family of operations $\max_u : D^2 \to D$, $u \in D$, defined such that

$$\max_u(a,b) = \begin{cases} u & \text{if } \max(a,b) \leq u \\ \max(a,b) & \text{otherwise} \end{cases}$$

Theorem 20 ([33]). Max Sol(Γ_L) *is tractable if* Γ_L *is invariant under* $\max_u$ *for some* $u \in D$*. Otherwise,* Max Sol(Γ_L) *is* **APX***-hard.*

Note that $Inv(\max_u)$ is generalised max-closed so the tractability part of Theorem 20 follows immediately from Theorem 18. The **APX**-hardness is proved by reductions from Max Independent Set and Max-E3SAT.

The Min Sol and Max AW Sol problems for clausal constraints have also been studied: define a new family of operations $\min_u : D^2 \to D$, $u \in D$, such that

$$\min_u(a,b) = \begin{cases} u & \text{if } \min(a,b) \geq u \\ \min(a,b) & \text{otherwise} \end{cases}$$

Theorem 21 ([33]). Min Sol(Γ_L) *is tractable if* Γ_L *is invariant under* $\min_u$ *for some* $u \in D$*. Otherwise,* Min Sol(Γ_L) *is* **APX***-hard.* Max AW Sol(Γ_L) *is tractable if* Γ_L *is simultaneously invariant under* max *and* min*. Otherwise,* Max AW Sol(Γ_L) *does not admit polynomial-time approximation schemes.*

The tractability proof for Min Sol is analogous to the proof of Theorem 18 while the tractability proof for Max AW Sol is based on supermodular maximisation [27, 39]. Note that we do not prove **APX**-hardness for Max AW Sol; the reason is that we are now forced to handle instances with negative optimal measure and **APX**-hardness and AP-reductions are only defined for problems with positive measure. However, it is still possible to rule out the existence of polynomial-time approximation schemes. For ordinary approximation problems, we say that a solution s is r-approximate if

$$\frac{\text{OPT}(I)}{r} \leq m(I,s) \leq \text{OPT}(I) \cdot r.$$

This does not work for problems with negative optima since in this case $\frac{\text{OPT}(I)}{r} \geq \text{OPT}(I) \cdot r$. With this in mind, we say that Π admits a Ptas if there exists an algorithm A satisfying the following property: for any instance I of Π and any rational value $r > 1$, $A(I,r)$ returns a solution s (in time polynomial in $|I|$) such that $m(I,s) \in [\text{OPT}(I)/r, r \cdot \text{OPT}(I)]$.

6 Binary symmetric relations

The complexity of $\textsc{Csp}(R)$ is known for every binary and symmetric relation R due to Hell and Nešetřil's celebrated result: $\textsc{Csp}(R)$ is **NP**-complete unless R is bipartite or contains a loop (and the problem is easily solvable in polynomial time). Such a dichotomy is not known for the Max Sol problem but there are some preliminary results by Jonsson *et al.* [34]. These results are presented next.

From now on, we view binary symmetric relations as undirected graphs in the obvious way where vertices denote domain elements and an edge (a, b) is present if and only if the tuples $(a, b), (b, a)$ are members of the relation. We do not restrict ourselves to reflexive or irreflexive graphs, i.e. each vertex may or may not have a loop. We begin by showing that we can concentrate on connected graphs.

Let $\mathcal{H} = \{H_1, \ldots, H_n\}$ be a set of connected graphs and let H be the disjoint union of these graphs. We are interested in the complexity of Max $\textsc{Sol}(H)$, given the complexities of the individual problems. Let $\mathcal{H}_i = \mathcal{H} \setminus \{H_i\}$. We say that H_i *extends* the set $\mathcal{H}_i$ if there exists an instance $I = (V, D, C, w)$ of Max $\textsc{Sol}(H_i)$ such that for all $1 \leq j \leq n, j \neq i$ it holds that $\textsc{opt}(I) > \textsc{opt}(I_j)$ where $I_j = (V, D_j, \{H_j(x, y) \mid H_i(x, y) \in C\}, w)$. We call I a *witness* to the extension.

Assume that for some $1 \leq i \leq n$, it holds that H_i does not extend $\mathcal{H}_i$. It is clear that for any connected instance $I = (V, D, C, w)$ of Max $\textsc{Sol}(H)$, we have $\textsc{opt}(I) = \textsc{opt}(I_j)$ for some j, where $I_j = (V, D_j, \{H_j(x, y) \mid H(x, y) \in C\}, w)$. Furthermore, since H_i does not extend $\mathcal{H}_i$, we know that we can choose this $j \neq i$. Let H' be the disjoint union of the graphs in $\mathcal{H}_i$. Then, $\textsc{opt}(I) = \textsc{opt}(I')$, where $I' = (V, D, \{H'(x, y) \mid H(x, y) \in C\}, w)$ is an instance of Max $\textsc{Sol}(H')$. For this reason, we may assume that every $H_i \in \mathcal{H}$ extends every graph in $\mathcal{H}_i$.

Lemma 22 ([34]). *Let $H_1, \ldots, H_n$ be graphs and H their disjoint union. If the problems* Max $\textsc{Sol}(H_i)$, $1 \leq i \leq n$ *are all tractable, then* Max $\textsc{Sol}(H)$ *is tractable. If* Max $\textsc{Sol}(H_i)$ *is* **NP***-hard and H_i extends the set $\{H_1, \ldots, H_{i-1}, H_{i+1}, \ldots, H_n\}$ for some i, then* Max $\textsc{Sol}(H)$ *is* **NP***-hard.*

Next, we need a couple of algorithmic results. Let $F = \{I_1, \ldots, I_k\}$ be a family of intervals on the real line. A graph G with $V(G) = F$ and $(I_i, I_j) \in E(G)$ if and only if $I_i \cap I_j \neq \emptyset$ is called an *interval graph*. If the intervals are chosen to be inclusion-free, G is called a *proper interval graph*.

Let $F_1 = \{I_1, \ldots, I_k\}$ and $F_2 = \{J_1, \ldots, J_l\}$ be two families of intervals on the real line. A graph G with $V(G) = F_1 \cup F_2$ and $(I_i, J_j) \in E(G)$ if and only if $I_i \cap J_j \neq \emptyset$ is called an *interval bigraph*. If the intervals in each family are chosen to be inclusion-free, G is called a *proper interval bigraph*.

Lemma 23 ([34]). *If H is a connected graph which is a proper interval graph or a proper interval bigraph, then* Max $\textsc{Sol}(H)$ *is polynomial time solvable.*

We will now present some complexity results for Max $\textsc{Sol}(H)$. Let H be an irreflexive graph such that $deg(H) \leq 2$. It is easy to see that H is the disjoint union of paths and cycles. By Lemma 22, we can without loss of generality

assume that H contains only one connected component. Every irreflexive path is a proper interval bigraph so Lemma 23 immediately gives a tractability result.

Proposition 24 ([34]). *If H is an irreflexive path, then* Max Sol(H) *is in* **PO**.

Assume now that H is a cycle. If H is an odd cycle, then Csp(H) and Max Sol(H) are **NP**-complete by Hell and Nešetřil's [22] result. If H is isomorphic to C_4, then H cannot be a max-core. One can see that the max-core of H is an irreflexive path so Max Sol(H) is in **PO** by Proposition 24 and Lemma 8. More generally, every irreflexive even cycle that is not a max-core has a max-core that is an irreflexive path. Thus, we can additionally assume that H is a max-core.

Proposition 25 ([34]). *Let H be a max-core which is isomorphic to an even cycle, i.e., C_{2k}. Assume a bipartition $V(H) = \{d_1, \ldots, d_k\} \cup \{d'_1, \ldots, d'_k\}$ of H with $d_1 < d_2 < \cdots < d_k$ and $d'_1 < d'_2 < \cdots < d'_k$ and, without loss of generality, assume that $d_k > d'_k$. We denote by $Pol_1(H)$ the set of unary polymorphisms of H. Let $F = \{\pi \in Pol_1(H) \mid \exists j \neq k : \pi(d_j) \neq d_j \vee \pi(d'_j) \neq d'_j\}$. If there exist non-negative constants $a_1, \ldots, a_{k-1}, a'_1, \ldots, a'_{k-1}$ such that for each $\pi \in Pol_1(H) \setminus F$, it is true that*

$$\sum_{i=1}^{k-1} \left(a_i \cdot d_i + a'_i \cdot d'_i\right) > \sum_{i=1}^{k-1} \left(a_i \cdot \pi(d_i) + a'_i \cdot \pi(d'_i)\right).$$

Then, Max Sol(H) *is* **NP**-*hard and, otherwise,* Max Sol(H) *is in* **PO**.

The proof of Proposition 25 largely builds on constructing unary relations via pp-formulas and exploiting complexity results for the *retraction* problem [15].

We now turn our attention to graphs H such that $|V(H)| \leq 4$. For $|V(H)| = 2$ there are only two (types) of max-cores, H_1 and H_2 in Figure 1. Max Sol(H_1) is

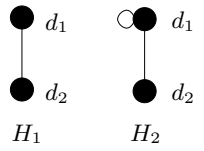

Fig. 1. Max-cores H_1 and H_2 for $V(H) = \{d_1, d_2\}$ where $d_1 < d_2$.

essentially the problem of finding the largest (heaviest) bipartition in a bipartite graph, which is in **PO**. Max Sol(H_2) is closely related to the Max Independent Set problem (in fact, it is exactly Max Independent Set for $d_1 = 0$ and $d_2 = 1$) and it is **NP**-hard (actually, **poly-APX**-complete when $d_1 = 0$, and **APX**-complete otherwise).

The complexity for 3-element graphs is completely determined in the next theorem. The main difficulty in proving the theorem is the tractability part

where the Critical Independent Set problem [1, 43] plays an important role. The hardness results can be obtained quite comfortably by reductions from the retraction problem and the Max Independent Set problem.

Theorem 26 ([34]). *There are six (types of) max-cores over* $\{d_1, d_2, d_3\}$ *where* $d_1 < d_2 < d_3$*, denoted* $H_1, \ldots, H_6$ *and shown in Figure 2.* Max Sol(H) *is* **NP***-hard for all of these except for* H_5*.* Max Sol(H_5) *is in* **PO** *if* $d_3 + d_1 \leq 2d_2$ *and* **NP***-hard otherwise.*

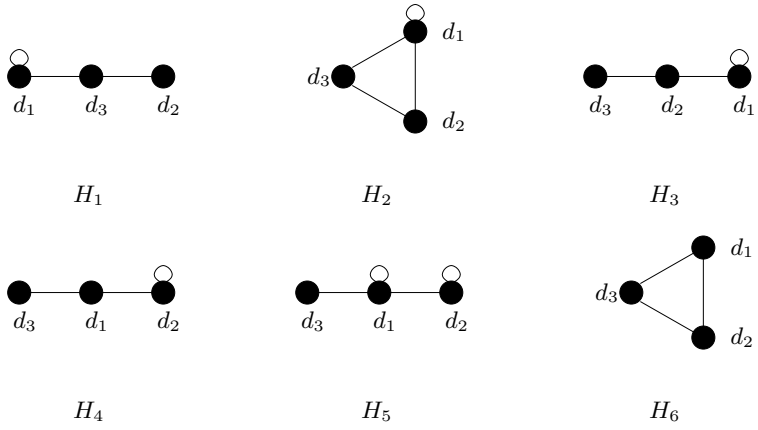

Fig. 2. Max-cores H_i for $|V(H)| = 3$.

We consider the graph H_5 closer since it highlights one of the difficulties with obtaining complexity classifications for Max Sol(Γ). Consider the graph $H = \{(2,0),(0,2),(0,0),(0,1),(1,0),(1,1)\}$ in Figure 3 and note that it is isomorphic to H_5 by setting $d_1 = 0$, $d_2 = 1$, and $d_3 = 2$. One consequence of Theorem 26 is

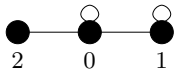

Fig. 3. A graph H for which Max Sol is in **PO**.

that Max Sol(H) is in **PO**. However, it is easy to see that H is a max-core but not an injective relation nor a relation that is invariant under a generalised max operation. The relation H is thus an example of a relation whose tractability cannot be explained in terms of the general results in Section 4. In fact, Max

Sol(H) is essentially the Critical Independent Set problem, which was shown to be in **PO** in [1, 43] by a rather clever algorithm.

If we instead consider the graph $H' = \{(3,0),(0,3),(0,0),(0,1),(1,0),(1,1)\}$ in Figure 4 (which is also isomorphic to H_5), then Theorem 26 tells us that Max Sol(H') is **NP**-hard. Hence, despite the striking similarity of the graphs H and H', they have different complexity with respect to the Max Sol problem. Moreover, it follows from Theorem 6 that the difference in complexity between Max Sol(H) and Max Sol(H') can be explained by analysing the set of polymorphisms of H and H', i.e., $Pol(H)$ and $Pol(H')$. In our opinion, this example, and

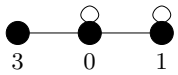

Fig. 4. A graph H' for which Max Sol is **NP**-hard.

in particular the fact that we have not (yet) been able to explain the tractability of Max Sol(H) in terms of properties of $Pol(H)$ indicate that it might be quite challenging to classify the complexity of Max Sol(Γ) over finite domains.

After this short digression we move on to the $|V| = 4$ case. The complexity of the $|V| = 4$ case is not completely known, but if we restrict ourselves to the vertex set $\{0,1,2,3\}$, then we have the following result:

Theorem 27 ([34]). *Let H be a max-core on $D = \{0,1,2,3\}$. Then,* Max Sol(H) *is in* **PO** *if H is an irreflexive path, and otherwise,* Max Sol(H) *is* **NP***-hard.*

The proof of Theorem 27 builds on the same ideas as the proof of Theorem 26.

7 Equations over groups

The complexity of solving equations over different algebraic structures is a very well-studied topic; we refrain from giving a long list of examples but simply remind the reader that solving linear equations is an instance of this problem. If we assign natural numbers to the elements of the structure, then it is obvious that we can also view such a problem as an instance of the Max Sol problem. In this section, we consider the Max Sol problem over group equations and this motivates the next definition.

Definition 28. *Let $\mathcal{G} = (G; +, -, 0_G)$ be a group (we use $+$ for the binary group operation, $-$ for inversion and 0_G for the identity element) and $g : G \to \mathbf{N}$ a function. The* Max Sol *problem over group equations is denoted by* Max Sol Eqn($\mathcal{G}$, g)*. An instance of* Max Sol Eqn($\mathcal{G}$, g) *is defined to be a triple (V, E, w) where,*

– *V is a set of variables,*
– *E is a set of equations of the form $w_1 + \ldots + w_k = 0_G$, where each w_i is either a variable, an inverted variable or a group constant, and*
– *w is a weight function $w : V \to \mathbf{N}$.*

The objective is to find an assignment $f : V \to G$ to the variables such that all equations are satisfied and the sum

$$\sum_{v \in V} w(v) \cdot g(f(v))$$

is maximised.

Note that the function g and the group $\mathcal{G}$ are not parts of the input so Max Sol Eqn($\mathcal{G}$, g) is parameterised by $\mathcal{G}$ and g.

Goldmann and Russell [18] have shown that solving systems of equations over non-Abelian groups is **NP**-hard. Thus, it is **NP**-hard to find feasible solutions to Max Sol Eqn($\mathcal{H}$, g) if $\mathcal{H}$ is non-Abelian, too. It is therefore sufficient to study Max Sol Eqn($\mathcal{H}$, g) where $\mathcal{H}$ is Abelian.

The main result is for groups of the form $\mathbf{Z}_p$ where p is prime. For a function $g : \mathbf{Z}_p \to \mathbf{N}$ we define the following two quantities,

$$g_{\max} = \max_{x \in \mathbf{Z}_p} g(x) \qquad \text{and} \qquad g_{\text{sum}} = \sum_{x \in \mathbf{Z}_p} g(x).$$

We are now ready to state the result.

Theorem 29 ([36]). *For every prime p and every function $g : \mathbf{Z}_p \to \mathbf{N}$,* Max Sol Eqn($\mathbf{Z}_p$, g) *is approximable within α where*

$$\alpha = \frac{p \cdot g_{\max}}{g_{\text{sum}}}.$$

Furthermore, for every prime p and every non-constant function $g : \mathbf{Z}_p \to \mathbf{N}$ Max Sol Eqn($\mathbf{Z}_p$, g) *is not approximable within $\alpha - \epsilon$ for any $\epsilon > 0$, unless $\mathbf{P} = \mathbf{NP}$.*

Note that if g is a constant function then every solution has the same measure. Obviously, an optimal can be found in polynomial time in this case. The inapproximability proof builds on the inapproximability of Max-Ek-Lin-G [26] combined with a series of reductions. The approximability result is obtained by an application of random sampling. This gives a randomised algorithm, which in expectation produces solutions of the required quality. A straightforward derandomisation of this algorithm is possible.

8 Maximal constraint languages

A *maximal constraint language* Γ is a constraint language such that $\langle\Gamma\rangle \subset R_D$, and if $R \notin \langle\Gamma\rangle$, then $\langle\Gamma \cup \{R\}\rangle = R_D$. That is, the maximal constraint languages

are the largest constraint languages that are not able to express all finitary relations over D. This implies, among other things, that there exists an operation f such that $\langle\Gamma\rangle = Inv(f)$ whenever Γ is a maximal constraint language [38]. The complexity of the $\text{CSP}(\Gamma)$ problem for all maximal constraint languages on domains $|D| \leq 3$ was determined in [8]. Moreover, it was shown in [8] that the only case that remained to be classified in order to extend the classification to all maximal constraint languages over a finite domain was the case where $\langle\Gamma\rangle = Inv(f)$ for binary commutative idempotent operations f. These constraint languages were finally classified by Bulatov in [5].

Theorem 30 ([5, 8]). *Let Γ be a maximal constraint language on an arbitrary finite domain D. Then, $\text{CSP}(\Gamma)$ is in* **P** *if $\langle\Gamma\rangle = Inv(f)$ where f is a constant operation, a majority operation, a binary commutative idempotent operation, or an affine operation. Otherwise, $\text{CSP}(\Gamma)$ is* **NP***-complete.*

We now present an approximability classification, from [32], of $\text{MAX SOL}(\Gamma)$ for all maximal constraint languages Γ over $|D| \leq 4$. Moreover, it is proved in [32] that the only cases that remain to be classified, in order to extend the classification to all maximal constraint languages over finite domains, are constraint languages Γ such that $\langle\Gamma\rangle = Inv(f)$ for a binary commutative idempotent operation f. It is also proved in [32] that if a certain conjecture regarding minimal clones generated by binary operations, due to Szczepara [40], holds, then the classification can be extended to capture also these last cases.

Theorem 31. *Let Γ be maximal constraint language on a finite domain D, with $|D| \leq 4$, and $\langle\Gamma\rangle = Inv(f)$.*

1. *If Γ is generalised max-closed or an injective constraint language, then* $\text{MAX SOL}(\Gamma)$ *is in* **PO***;*
2. *else if f is an affine operation, a constant operation different from the constant 0 operation, or a binary commutative idempotent operation satisfying $f(0, b) > 0$ for all $b \in D \setminus \{0\}$ (assuming $0 \in D$); or if $0 \notin D$ and f is a binary commutative idempotent operation or a majority operation, then* $\text{MAX SOL}(\Gamma)$ *is* **APX***-complete;*
3. *else if f is a binary commutative idempotent operation or a majority operation, then* $\text{MAX SOL}(\Gamma)$ *is* **poly-APX***-complete;*
4. *else if f is the constant 0 operation, then finding a solution with non-zero measure is* **NP***-hard;*
5. *otherwise, finding a feasible solution is* **NP***-hard.*

Moreover, if Conjecture 131 from [40] holds, then the results above hold for arbitrary finite domains D.

The proof of the preceding theorem consists of a careful analysis of the approximability of $\text{MAX SOL}(\Gamma)$ for all maximal constraint languages Γ such that $\langle\Gamma\rangle = Inv(f)$, where f is one of the types of operations in Theorem 30.

9 Homogeneous constraint languages

In this section, we describe a classification result from [32] on the complexity of Max Sol when the constraint language is *homogeneous*. A constraint language is called homogeneous if every *permutation relation* is contained in the language.

Definition 32. *A binary relation R is a* permutation relation *if there is a permutation $\pi : D \to D$ such that*

$$R = \{(x, \pi(x)) \mid x \in D\}.$$

Let Q denote the set of all permutation relations on D. The complexity classification of Max Sol(Γ) when $Q \subseteq \Gamma$ from [32] provide the exact borderlines between tractability, **APX**-completeness, **poly-APX**-completeness, and **NP**-hardness of finding a feasible solution. Due to space constraints we only describe the borderline between tractability (i.e., membership in **PO**) and **NP**-hardness. This classification is described in Theorem 33.

Dalmau completely classified the complexity of Csp(Γ) when Γ is a homogeneous constraint language [14], and this classification relies heavily on the structure of homogeneous algebras. An algebra is called *homogeneous* if and only if every permutation on its universe is an automorphism of the algebra. For a formal definition and further information on the properties of homogeneous algebras we refer the reader to [41].

The approximability classification of Max Sol(Γ) when Γ is a homogeneous constraint language uses the same approach as in [14], namely, the inclusion structure of homogeneous algebras is exploited. The (only) tractable class of Max Sol(Γ), over homogenous constraint languages, can be characterised by the presence in $Pol(\Gamma)$ of a discriminator operation t (as defined in Section 4.1).

Theorem 33 ([32]). *Let Γ be a homogeneous constraint language. If a discriminator operation t is in $Pol(\Gamma)$, then* Max Sol(Γ) *is in* **PO**. *Otherwise,* Max Sol(Γ) *is* **NP***-hard.*

Since a constraint language Γ is injective if and only if $\Gamma \subseteq Inv(t)$, the tractability part of the theorem follows from Theorem 14. The hardness part of the theorem is proved by reductions from variants of the Max Independent Set problem and the Max Sol Eqn(G, g) problem, as defined in Section 7.

As a direct consequence of Theorem 33 we get that the class of injective relations, as defined in Section 4.1, is a maximal tractable class for Max Sol(Γ). That is, if we add a single relation which is not an injective relation to the class of all injective relations, then the problem is no longer in **PO** (unless **P** = **NP**).

10 Outlook

In this section, we describe the relationship between the Max Sol problem and some other frameworks for optimisation problems.

10.1 Relation to Valued CSPs

We begin by giving a simplified account of the VCSP (valued CSP) framework studied in, e.g., [9, 10]. The VCSP framework involves two types of constraints: crisp constraints (which must be satisfied) and soft constraints (which are satisfied by any assignment, but different assignments may generate different costs). More formally, a soft constraint (or valued constraint) is a pair (σ, φ) where $\sigma = (x_1, \ldots, x_k)$ is a k-tuple of variables (the constraint scope) and φ is a k-ary cost function from D to $\mathbb{Z}$. The cost of the assignment $(a_1, \ldots, a_k)$ is given by $\varphi(a_1, \ldots, a_k)$. The objective is to find a solution (satisfying all crisp constraints) that minimise (or maximise) the total cost of all soft constraints.

Just as for the ordinary Csp problem, much effort has been put into studying the complexity of the VCSP problem for various constraint languages. The constraint language can be viewed as consisting of two parts: the crisp constraint language Γ (a set of relations) and the valued constraint language Γ_V (a set of cost functions). The $\mathrm{VCSP}(\Gamma, \Gamma_V)$ problem can now be defined as follows:

Definition 34. *A* $\mathrm{VCSP}(\Gamma, \Gamma_V)$ *instance is a tuple* (V, C, C_V, D)*, where* V *is a set of variables,* C *is a set of crisp constraints (whose constraint relations are in* Γ*),* C_V *is a set of valued constraints (whose cost functions are in* Γ_V*), and* D *is the domain. The objective is to find an assignment* $f : V \to D$ *such that all constraints in* C *are satisfied and the sum*

$$\sum_{(\sigma,\varphi)\in C_V} \varphi(f(\sigma[1]), \ldots, f(\sigma[i]))$$

is minimised.

We denote the corresponding problem where the objective is to find the solution that maximises the sum above by Max-$\mathrm{VCSP}(\Gamma, \Gamma_V)$. Note that the definitions of VCSP problems in [9, 10] are much more sophisticated and elegant than our simplified account above. We also remark that, in order to comply with the existing literature on VCSP problems, we do not assume that the domain and the constraint languages Γ and Γ_V are finite.

It is easy to see that instances of Max Sol(Γ), Min Sol(Γ), and Max AW Sol(Γ) problems can be seen as instances of VCSP problems.

Proposition 35.

- *Instances of* Max Sol(Γ) *can be seen as instances of* Max-$\mathrm{VCSP}(\Gamma, \Gamma_V)$ *where* Γ_V *consists of a single unary cost function* φ*, defined such that* $\varphi(d) = d$ *for all* $d \in D$*.*
- *Instances of* Min Sol(Γ) *can be seen as instances of* $\mathrm{VCSP}(\Gamma, \Gamma_V)$ *where* Γ_V *consists of a single unary cost function* φ*, defined such that* $\varphi(d) = d$ *for all* $d \in D$*.*
- *Instances of* Max AW Sol(Γ) *can be seen as instances of* Max-$\mathrm{VCSP}(\Gamma, \Gamma_V)$ *where* Γ_V *consists of two unary cost functions* φ *and* $\overline{\varphi}$*, such that* $\varphi(d) = d$ *for all* $d \in D$*, and* $\overline{\varphi}(d) = -d$ *for all* $d \in D$*.*

Variable weights can be handled in the VCSP *setting by repeated applications of the unary cost function* φ *(and* $\overline{\varphi}$ *in the case of negative weights).*

10.2 Relation to Minimum Cost Homomorphism

The Minimum Cost Homomorphism problem (denoted Min Hom(H)) has recently received a lot of attention [19–21]. The problem Min Hom(H) can be defined as follows:

Definition 36. *An instance of* Min Hom(H) *is a graph G together with a weight function $w(x, y) : V(G) \times V(H) \to \mathbb{N}$. The objective is to find a homomorphism $h : V(G) \to V(H)$ such that the sum*

$$\sum_{v \in V(G)} w(v, h(v))$$

is minimised.

An instance of the Min Hom(H) problem can also be seen as an instance of the VCSP(Γ, Γ_V) problem, where Γ consists of the single binary relation H and Γ_V is a set of unary cost functions, defined such that $\varphi_v(d) = w(v, d)$ for all $v \in V(G)$ and $d \in V(H)$. Note that Γ_V, as defined here, is not finite. We note in passing that an analogous result to Theorem 6 holds for Min Hom(H).

Proposition 37. *Let H be a graph and $\Gamma' \subseteq \langle H \rangle$ finite. Then,* Min Hom(Γ') *is S-reducible to* Min Hom(H).

A dichotomy is known for the complexity of Min Hom(H) in the case when H is an undirected graph.

Theorem 38 ([19]). *If each component of H is a proper interval graph or a proper interval bigraph, then the problem* Min Hom(H) *is in* **PO**. *In all other cases,* Min Hom(H) *is* **NP**-*hard.*

A list extension of the Max AW Sol(H) problem (denoted List Max AW Sol(H)) is studied in [34]. The Max AW Sol(H) problem is extended by introducing lists, $\{L(v) \subseteq V(H) \mid v \in V(G)\}$, and requiring that any solution must assign to v one of the vertices in $L(v)$. It is not hard to realise that the List Max AW Sol(H) problem is a restriction of the Min Hom(H) problem. One of the results of [34] is a complexity classification of List Max AW Sol(H) for undirected graphs H.

Theorem 39 ([34]). *Let H be an undirected graph with loops allowed. Then* List Max AW Sol(H) *is solvable in polynomial time if all components of H are proper interval graphs or proper interval bigraphs. Otherwise* List Max AW Sol(H) *is* **NP**-*hard.*

The preceding result, together with the complexity classification of Min Hom(H) from [19], gives us the following corollary.

Corollary 40. *Let H be an undirected graph with loops allowed. Then,* List Max AW Sol(H) *is polynomial-time equivalent to* Min Hom(H).

11 Open questions

The long-term goal for this line of research is, of course, to completely classify the approximability of Max Sol for all finite constraint languages. However, this is probably a hard problem since not even a complete classification for the corresponding decision problem Csp is known. A more manageable task would be to completely classify Max Sol for constraint languages over small domains (say, of size 3 or 4). For size 3, this has already been accomplished for Csp [6] and Max Csp [31]. Another obvious open problem is to classify the complexity of Max Sol(Γ) for the remaining maximal constraint languages Γ described in Section 8. The known results for the complexity of Max Sol suggest the following conjecture:

Conjecture 41. For every finite constraint language Γ, one of the following holds:

1. Max Sol(Γ) is in **PO**;
2. Max Sol(Γ) is **APX**-complete;
3. Max Sol(Γ) **poly-APX**-complete;
4. it is **NP**-hard to find a non-zero solution to Max Sol(Γ); or
5. it is **NP**-hard to find any solution to Max Sol(Γ).

If this conjecture is true, then there does not exist any constraint language Γ such that Max Sol(Γ) has a polynomial-time approximation scheme (Ptas) but Max Sol(Γ) is not in **PO**. However, if one impose simultaneous restrictions on the allowed constraint types and the way constraints are applied to variables (instead of only restricting the allowed constraint types), then the situation changes. Consider for example Max Independent Set (and, equivalently, Max Ones($\{(0,0),(1,0),(0,1)\}$)): the unrestricted problem is **poly-APX**-complete and not approximable within $O(n^{1-\epsilon})$, $\epsilon > 0$ (unless **P**=**NP**) [44], but the problem restricted to planar instances admits a Ptas [3]. One may ask several questions in connection with this: For which constraint languages does Max Sol admit a Ptas on planar instances? Or more generally: under what restrictions on variable scopes does Max Sol(Γ) admit a Ptas?

The investigation of the complexity of Max Sol(Γ) when Γ is a graph [34] indicates that giving a complete complexity classification of Max Sol(Γ) for every fixed constraint language Γ is probably harder than first anticipated. In particular, the tractable class for the Max Sol problem identified in [34] (and described in Section 6) depends very subtly on the values of the domain elements and no characterisation of this tractable class in terms of polymorphisms is known. Hence, this tractable class seems to be of a different flavour compared to the other tractable classes for the Max Sol problem [32, 33, 35]. On the other hand, there is a complete classification for the complexity of the arbitrary weighted list version of the problem, List Max AW Sol(Γ), in the important case where Γ is an undirected graph.

Interestingly, for undirected graphs H the borderline between tractability and **NP**-hardness for List Max AW Sol(H) coincide exactly with Gutin *et al.*'s [19] recent complexity classification of Min Hom(H). This is surprising,

since the Min Hom(H) problem is much more expressive than the List Max AW Sol(H) problem, and we were expecting graphs H such that Min Hom(H) were **NP**-hard and List Max AW Sol(H) were in **PO**. Moreover, it is not hard to prove that Min Hom(Γ) over Boolean constraint languages Γ is in **PO** if Γ is width-2 affine, or if Γ is Horn and dual-Horn; and that Min Hom(Γ) is **NP**-hard for all other Boolean constraint languages Γ. This borderline between **PO** and **NP**-hardness coincides with the borderline between **PO** and **NP**-hardness for Max AW Sol(Γ) over Boolean constraint languages Γ proved in [30]. The obvious question raised by these results is how far can we extend the correspondence in complexity between List Max AW Sol(Γ) and Min Hom(Γ)? More specifically, is it the case that List Max AW Sol(Γ) is in **PO** (**NP**-hard) if and only if Min Hom(Γ) is in **PO** (**NP**-hard) for arbitrary finite constraint languages Γ?

Acknowledgments

The authors thank Fredrik Kuivinen and Johan Thapper for making important contributions to this survey. We also thank the anonymous referee for providing some very useful comments. Peter Jonsson is partially supported by the *Center for Industrial Information Technology* (CENIIT) under grant 04.01, and by the *Swedish Research Council* (VR) under grant 621-2003-3421. Gustav Nordh is partially supported by the *Swedish-French Foundation*, and by the *National Graduate School in Computer Science* (CUGS), Sweden.

References

1. A. Ageev. On finding critical independent and vertex sets. *SIAM J. Discrete Math.*, 7(2):293–295, 1994.
2. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti Spaccamela, and M. Protasi. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer, 1999.
3. B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41:153–180, 1994.
4. M. Bodirsky and J. Nešetřil. Constraint satisfaction with countable homogeneous templates. *Journal of Logic and Computation*, 16(3):359–373, 2006.
5. A. Bulatov. A graph of a relational structure and constraint satisfaction problems. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS 2004)*, pages 448–457, 2004.
6. A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006.
7. A. Bulatov, P. Jeavons, and A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005.
8. A. Bulatov, A. Krokhin, and P. Jeavons. The complexity of maximal constraint languages. In *Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC 2001)*, pages 667–674, 2001.

9. D. Cohen, M. Cooper, and P. Jeavons. An algebraic characterisation of complexity for valued constraint. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP 2006)*, pages 107–121, 2006.
10. D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. The complexity of soft constraint satisfaction. *Artificial Intelligence*, 170(11):983–1016, 2006.
11. N. Creignou, M. Hermann, A. Krokhin, and G. Salzer. Complexity of clausal constraints over chains. *Theory Comput. Syst.*, 42(2):239–255, 2008.
12. N. Creignou, S. Khanna, and M. Sudan. *Complexity classifications of Boolean constraint satisfaction problems*. SIAM, Philadelphia, 2001.
13. P. Crescenzi, R. Silvestri, and L. Trevisan. On weighted vs unweighted versions of combinatorial optimization problems. *Inf. Comput.*, 167(1):10–26, 2001.
14. V. Dalmau. A new tractable class of constraint satisfaction problems. *Annals of Mathematics and Artificial Intelligence*, 44(1–2):61–85, 2005.
15. T. Feder, P. Hell, and J. Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19:487–505, 1999.
16. T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1999.
17. À. Gil, M. Hermann, G. Salzer, and B. Zanuttini. Efficient algorithms for constraint description problems over finite totally ordered domains. In *Proceedings of Automated Reasoning, Second International Joint Conference (IJCAR 04)*, pages 244–258, 2004.
18. M. Goldmann and A. Russell. The complexity of solving equations over finite groups. In *IEEE Conference on Computational Complexity*, pages 80–86, 1999.
19. G. Gutin, P. Hell, A. Rafiey, and A. Yeo. A dichotomy for minimum cost graph homomorphisms. *European J. Combin.*, 29(4):900–911, 2008.
20. G. Gutin, A. Rafiey, and A. Yeo. Minimum cost and list homomorphisms to semicomplete digraphs. *Discrete Applied Mathematics*, 154(6):890–897, 2006.
21. 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.
22. P. Hell and J. Nešetřil. On the complexity of H-colouring. *Journal of Combinatorial Theory B*, 48:92–110, 1990.
23. R. Hähnle. Complexity of many-valued logics. In *Proceedings of the 31st IEEE International Symposium on Multiple-valued Logic (ISMVL 01)*, pages 137–148, 2001.
24. D. Hochbaum, N. Megiddo, J. Naor, and A. Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Mathematical Programming*, 62:69–84, 1993.
25. D. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. Comput.*, 23(6):1179–1192, 1994.
26. J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.
27. S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM*, 48(4):761–777, 2001.
28. P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44:527–548, 1997.

29. P. Jeavons and M. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79:327–339, 1996.
30. P. Jonsson. Boolean constraint satisfaction: complexity results for optimization problems with arbitrary weights. *Theoretical Computer Science*, 244(1-2):189–203, 2000.
31. P. Jonsson, M. Klasson, and A. Krokhin. The approximability of three-valued Max CSP. *SIAM J. Comput.*, 35(3):1329–1349, 2006.
32. P. Jonsson, F. Kuivinen, and G. Nordh. Max Ones generalised to larger domains. *SIAM J. Comput.*, 38(1):329–365, 2008.
33. P. Jonsson and G. Nordh. Generalised integer programming based on logically defined relations. In *Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science (MFCS 2006)*, pages 549–560, 2006.
34. P. Jonsson, G. Nordh, and J. Thapper. The maximum solution problem on graphs. In *Proceedings of the 32nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2007)*, pages 228–239, 2007.
35. S. Khanna, M. Sudan, L. Trevisan, and D. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6):1863–1920, 2001.
36. F. Kuivinen. Tight approximability results for the maximum solution equation problem over Z_p. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS 2005)*, pages 628–639, 2005.
37. R. Pöschel and L. Kaluznin. *Funktionen- und Relationenalgebren*. DVW, Berlin, 1979.
38. I. Rosenberg. Minimal clones I: the five types. In L. Szabó and Á. Szendrei, editors, *Lectures in Universal Algebra*. North-Holland, 1986.
39. A. Schrijver. A combinatorial algorithm for minimizing submodular functions in polynomial time. *Journal of Combinatorial Theory B*, 80:346–355, 2000.
40. B. Szczepara. *Minimal clones generated by groupoids*. PhD thesis, Université de Montréal, 1996.
41. Á. Szendrei. *Clones in Universal Algebra*, volume 99 of *Séminaires de Mathématiques Supérieures*. University of Montreal, 1986.
42. G. Woeginger. An efficient algorithm for a class of constraint satisfaction problems. *Operations Research Letters*, 30(1):9–16, 2002.
43. C. Zhang. Finding critical independent sets and critical vertex subsets are polynomial problems. *SIAM J. Discrete Math.*, 3(3):431–438, 1990.
44. D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th ACM Symposium on Theory of Computing (STOC 2006)*, pages 681–690, 2006.