Official URL:

https://doi.org/10.1007/s10472-017-9566-6

To cite this version:

Jguirim, Wafa and Naanaa, Wady and Cooper, Martin C. *A polynomial relational class of binary CSP.* (2018) Annals of Mathematics and Artificial Intelligence, 83 (1). 1-20. ISSN 1012-2443 .

A polynomial relational class of binary CSP

Wafa Jguirim · **Wady Naanaa** ·
Martin C. Cooper

Abstract Finding a solution to a constraint satisfaction problem (CSP) is known to be an NP-hard task. Considerable effort has been spent on identifying tractable classes of CSP, in other words, classes of constraint satisfaction problems for which there are polynomial time recognition and resolution algorithms. In this article, we present a relational tractable class of binary CSP. Our key contribution is a new ternary operation that we name mjx. We first characterize mjx-closed relations which leads to an optimal algorithm to recognize such relations. To reduce space and time complexity, we define a new storage technique for these relations which reduces the complexity of establishing a form of strong directional path consistency, the consistency level that solves all instances of the proposed class (and, indeed, of all relational classes closed under a majority polymorphism).

1 Introduction

Many real-world problems may be formulated by means of constraints on time, on space or more generally on resources. Planning, scheduling and resource allocation are just a few among many problems that involve reasoning about constraints. Such problems are designated by the general term constraint satisfaction problem (CSP) and are highly combinatorial, because their solutions are to be found among a huge set of combinations. In terms of complexity theory, solving a CSP is, in general, an NP-complete task. Nonetheless, many real word CSPs have specific properties that make them recognizable and solvable in polynomial time. Thus, despite the NP-completeness of the CSP, many of its instances fall into *tractable*

W. Jguirim
Faculty of Sciences, University of Monastir, Monastir 5000 Tunisie
E-mail: jguirimwafa06@gmail.com

W. Naanaa
Faculty of Sciences, University of Monastir, Monastir 5000 Tunisie
E-mail: wady.naanaa@fsm.rnu.tn

M. Cooper
IRIT, University of Toulouse III, 31062 Toulouse.
E-mail: cooper@irit.fr

classes that can be recognized and solved in polynomial time. Tractable CSP classes fall into three categories: *structural* classes, *relational* classes and *hybrid* classes [CC16]. Structural classes are based on the topology of constraint networks, which often have specific properties, especially when they originate from real-world problems. Acyclic networks [Fre85] and bounded tree-width networks [GLS00] are two tractable structural classes. Indeed, in the case of bounded-arity constraints, it is known that all tractable structural classes have bounded treewidth [Gro07].

On the other hand, the idea behind relational CSP classes is to limit the constraint semantics, giving rise to the notion of *constraint language*. Relational CSP classes are identified by pointwise closure operations, known as *polymorphisms*, since the existence of a polymorphism is known to be a necessary condition for tractability [Jea98]. Max-closed CSPs [JC95] and median-closed CSPs [JCC98, DBH99] are two well known tractable relational classes of CSPs which generalize, respectively, HornSAT and 2-SAT. On the more abstract level of universal algebra, it is the identities satisfied by a polymorphism f that guarantees tractability [BJK05], such as $f(x,x,y) = f(x,y,x) = f(y,x,x) = x$ for majority polymorphisms f [JCC98].

More recently, diverse tractable classes have been discovered which simultaneously rely on structural and relational properties, giving rise to so-called *hybrid* tractable CSP classes. The Broken-Triangle Property class [CJS10, CMTZ14] and the bounded rank CSP class [Naa13] are just two examples of hybrid classes.

In this context, we propose a new relational class of binary CSPs based on a novel majority operator, which we call mjx. The proposed operator makes the natural choice of returning the maximum of its three arguments in case they are all different. From a theoretical point of view, recent work [BKW12, BK14] has made important steps towards a proof of the Feder-and-Vardi conjecture, which states that every finite relational class is either polynomial or NP-complete [FV98]. Nonetheless, from a practical point of view, there is much to be done in order to identify tractable CSP classes that are useful in real-world applications. The aim of this paper is to contribute to the identification of constraint languages that possess a natural semantics which makes them useful in practice.

If Γ is a tractable constraint language, then instances which fall into the corresponding relational class may be relatively rare in practice. Nevertheless, given any instance I, the sub-instance consisting of those constraints which belong to Γ provides a potentially useful polynomial-time relaxation of I. Such relaxations can provide useful information during search: we can draw a parallel, for example, with the linear programming relaxation of integer programming instances or global constraints (such as the All-Different constraint) in CSP instances [vHK06]. To provide a useful relaxation, a language Γ must be have an efficient recognition algorithm as well as en efficient consistency-checking algorithm [CC16].

The rest of the paper is organized as follows: Section 2 introduces the notions, definitions and notation necessary for the description of our new tractable class. We also give, by way of examples, some relations which are closed under mjx. Section 3 presents an alternative characterization of mjx-closed binary relations that enables an optimal identification of these relations for a given domain ordering. In Section 4, we show that mjx-closed binary CSP instances can be solved, in polynomial time, by establishing directional path consistency and a weak form of arc consistency. We present an algorithm which efficiently enforces this level of consistency. This algorithm optimizes directional path consistency owing to new computations of intersection and composition operations, which are tailored for mjx-closed binary relations. Indeed, contrary to existing general-purpose algorithms [BRYZ05], the proposed algorithms for computing the intersection and composition of mjx-closed binary relations have a linear complexity $O(d)$, where d is the size of the largest domain. Finally, a conclusion and some pointers to future work are given in Section 5.

2 Definitions and notation

We first introduce some definitions and notation related to the CSP framework and tractable CSP classes.

Definition 1 A constraint satisfaction problem (CSP) is defined by a triple (X, D, C) where:

- $X = \{x_1, ..., x_n\}$ is a finite set of n variables.
- D is a finite value domain.
- $C = \{c_1, ..., c_m\}$ is a set of constraints. Each constraint c_i is a pair $(S(c_i), R(c_i))$ where
  - $S(c_i) \subseteq X$ is the scope of c_i
  - $R(c_i) \subseteq D^{|S(c_i)|}$ is the relation specifying the tuples allowed by c_i.

The arity of a constraint c_i refers to the cardinality, or size of its scope, that is $|S(c_i)|$. A binary constraint is defined on two variables. A binary constraint network has only binary and unary constraints. A *partial instantiation* is a set of elementary instantiations to distinct variables, where an *elementary* instantiation is an ordered pair (x_i, v_i), which assigns value $v_i \in D_i$ to variable x_i, where $D_i \subseteq D$ is a unary relation that defines the unique unary constraint whose scope is $\{x_i\}$. An instantiation that assigns a value to every variable is said to be *complete*. A partial instantiation I satisfies a constraint c_i if $S(c_i) \subseteq S(I)$ and $(I \downarrow S(c_i)) \in R(c_i)$, where $S(I)$ designates the variables to which I assigns values and $I \downarrow S(c_i)$ designates the tuple of values assigned, by I, to the variables of $S(c_i)$. A partial instantiation that satisfies all the constraints is said to be *consistent*. A *solution* is a consistent and complete instantiation. Solving the decision version of CSP consists in determining whether a solution exists.

A CSP instance may possess a limited form of consistency, called *local consistency*, which offers the advantage of an easy calculation of limited-size consistent instantiations. There are many local consistency levels, that can be distinguished by means of a single parameter as follows [Fre85]:

Let $P = (X, D, C)$ be a CSP instance and let k be an integer between 1 and $|X|$.

- A consistent instantiation I of any $k-1$ variables of P is said to be k-*consistent* relative to an additional variable x_k if there exists $v_k \in D_k$ such that $I \cup \{(x_k, v_k)\}$ is consistent.
- P is said to be k-*consistent* if every consistent instantiation of any $k-1$ variables is k-consistent relative to every additional variable.

If P is i-consistent, for all $i : 1, \ldots, k$ then it is said to be *strongly* k-consistent, and if it is strongly $|X|$-consistent then it is called *globally* consistent. The most used levels of local consistency are *node*, *arc* and *path* consistencies, which stand respectively for 1-, 2- and 3-consistency.

Enforcing k-consistency, for some k, is achieved by removing some value combinations from the relations defining the constraints. The worst-case time complexity of enforcing k-consistency is $O(|X|^k|D|^k)$ [Coo89]. However, this process can be lightened by considering a weak form of k-consistency called *directional k-consistency* [DP87]. Assume that the variables of P are totally ordered by $\prec$. Then P is *directional* k-consistent with respect to $\prec$ if every consistent instantiation I of any $k-1$ variables is k-consistent relative to every variable that comes, in the ordering, after all variables instantiated by I. P is *strongly* directional k-consistent if it is directional i-consistent with respect to $\prec$, for all $i : 1, \ldots, k$. As mentioned above, directional k-consistency is a weak form of k-consistency in the sense that k-consistency entails directional k-consistency but the converse is not true.

Example 1 Some of the above-defined local consistency levels can be illustrated by considering a small CSP instance on three variables, x_1, x_2, x_3, each with value domain $D = \{0, 1, 2\}$. The instance also involves three binary constraints defined by $x_1 < x_2$, $x_1 \leq x_3$ and $x_2 \leq x_3$. Since the three value domains are not empty, any empty instantiation is node consistent relative to any variable of the instance. Thus, the whole instance is node consistent. Nonetheless, the instance is not arc consistent, since the consistent instantiation $\{(x_1, 2)\}$ is not arc consistent relative to variable x_2. The instance is not path consistent either, since the consistent instantiation $\{(x_1, 1), (x_3, 1)\}$ is not path consistent relative to x_2. Now, if we refer to the variable ordering $x_1 \prec x_2 \prec x_3$, we can verify that every consistent instantiation involving x_1 and x_2 simultaneously is path consistent relative to x_3 (such an instantiation can be consistently extended using $(x_3, 2)$). This means that the instance is directional path consistent with respect to $x_1 \prec x_2 \prec x_3$. However, the instance is not directional arc consistent, whatever the variable ordering is. This is due to the constraint $x_1 < x_2$, which means that there exists a consistent instantiation of x_1, resp. x_2, which is not arc consistent relative to x_2, resp. x_1.

We also need to recall some definitions and concepts used in relational (i.e. language) tractability.

Definition 2 A constraint language Γ is a set of relations over some given domain $D(\Gamma)$. We denote by CSP(Γ) the set of all instances $(X, D(\Gamma), C)$, such that for all $(S, R) \in C$, $R \in \Gamma$.

A finite constraint language Γ is tractable if there is a polynomial algorithm that solves all the instances of CSP(Γ). An infinite constraint language Γ is tractable if every finite subset of the language is tractable.

Definition 3 Consider an operator $\phi : D^k \to D$. A binary relation R over D is ϕ-closed if, for all (a_1, a'_1),…,$(a_k, a'_k) \in D^2$, we have (a_1, a'_1),…, $(a_k, a'_k) \in R \Rightarrow (\phi(a_1, \ldots, a_k), \phi(a'_1, \ldots, a'_k)) \in R$. A constraint language Γ is closed under ϕ if all relation of Γ are ϕ-closed.

In the following, we introduce our new operator, called mjx. It is a ternary operator that belongs to the class of majority operators. Such a class contains all the ternary operators that belong to the family of near-unanimity operators [JCC98]. Let $\phi : D^3 \to D$ be a ternary operation defined on a domain D. ϕ is a majority operator if, for all $a, b \in D$, we have

$$\phi(b, a, a) = \phi(a, b, a) = \phi(a, a, b) = a$$

Definition 4 Consider a domain D that is totally ordered, The majority operator mjx is defined on D as follows:

$$\mathrm{mjx}(a, b, c) \stackrel{\text{def}}{=} \begin{cases} a & \text{if} \quad a = b \ \vee \ a = c \\ b & \text{if} \quad a \neq b \ \wedge \ b = c \\ \max(a, b, c) & \text{otherwise} \end{cases}$$

A relation R is mjx-closed if we have

$$(a, a'), (b, b'), (c, c') \in R \quad \Rightarrow \quad (\mathrm{mjx}(a, b, c), \mathrm{mjx}(a', b', c')) \in R$$

Let us recall a theorem stating that the set of CSP instances defined over a language which is closed under a majority operator is a tractable CSP class.

Theorem 1 *[JCC98, BKW12] Let Γ be any set of relations over a finite domain, D. If Γ is closed under a majority operation then CSP(Γ) is solvable in polynomial time.*

Firstly, if an r-ary relation R, with $r \geq 2$, is closed under a majority operation ϕ, then any r-ary constraint (S, R) is equivalent to the conjunction of a set of $\binom{r}{2}$ binary constraints, each defined via a binary relation obtained by projecting R on two of its components. Moreover, all these binary relations are also closed under ϕ. Secondly, all binary instances consisting of only ϕ-closed constraints can be solved by establishing strong path consistency. It follows that closure of all relations by a majority operator ϕ is a sufficient condition that allows CSP instances to be solved in polynomial time [JCC98, BKW12].

Corollary 1 *Let $I \in CSP(\Gamma)$. If Γ is closed under* mjx *then P can be solved in polynomial time.*

Proof mjx is a majority operator since it verifies the property of majority operators:

$$\forall a, b \quad \mathrm{mjx}(a, a, b) = \mathrm{mjx}(a, b, a) = \mathrm{mjx}(b, a, a) = a$$

Tractability then follows from Theorem 1.

The proposed operator, mjx, is a natural choice among all majority operators as it returns the maximum of its arguments whenever they are all different. It is important to know whether there are mjx-closed relations that might occur in practice. In the following, we give some examples of such relations. The fact that these relations are closed under mjx follows from the characterization of mjx-closed relations that will be given in Section 3.

Example 2 Every unary relation $(x \in A) \subset D$ is mjx-closed. Every binary relation over Boolean domains is mjx-closed. Therefore, the class of CSP instances whose relations are mjx-closed can be considered as a generalization of 2-SAT. The CRC ("connected row convex") CSP is another generalization of 2-SAT that is incomparable with mjx-closed CSP.

Example 3 Let D be a totally-ordered domain. Let $f : D \to D$ be a monotonically decreasing function, that is, a function that verifies $u < v \Rightarrow f(u) \geq f(v)$. The following binary relations are mjx-closed, where x, y are variables and a, b, c are constants:

$$
\begin{gathered}
x \geq f(y) \\
x + y \geq a \\
(x \geq a) \vee (y \geq b) \\
x = y + c \\
(x = y + c) \vee (x \geq f(y)) \\
(x = a) \vee (y = b) \\
(x = a) \vee (x \geq f(y)) \\
(x = a) \vee (y = b) \vee (x \geq f(y)) \\
((x = a) \wedge (y = b)) \vee (x \geq f(y))
\end{gathered}
$$

In the last four examples, the values a and b could represent default values. For example, let x be the time at which an action starts in a planning problem, $x = a$ could represent the fact that we do not execute the action.

3 Identifying mjx-closed relations

The most intuitive way to check whether a relation R over a finite domain D is mjx-closed consists in testing all possible combinations of three elements of R. More precisely, we need to check the mjx-closure of each set of three pairs in R. This simple method requires d^6 tests, where $d = |D|$, which could be a huge number if d is large.

The following section describes a more efficient method for identifying mjx-closed relations.

3.1 Characterizing mjx-closed relations

In this section, we give necessary and sufficient conditions for closure under mjx. Let R be a binary relation over a finite D. A possible representation of R consists in using a $d \times d$ Boolean matrix, where $d = |D|$, whose rows and columns are indexed by the elements of D. For instance, Figure 1-left depicts a Boolean matrix which represents the binary relation $(|x-y| = 2) \vee (x+y > 4)$ over $D = \{0, 1, 2, 3\}$. Accordingly, we also use R to denote the Boolean matrix associated with R. The matrix R^* is derived from R by removing rows and columns containing only zeros. A simple space-efficient way to deduce R^* from R is to resort to two length-d arrays that contain, respectively, the indices of the non-zero rows and the indices of the non-zero columns of R. In addition, such arrays are also necessary for establishing arc consistency.

Proposition 1 *Let R be a binary relation such that $R{=}R^*$ (R does not contain rows or columns of zeros). If R is* mjx*-closed then $\forall a, a', b, b'$,*

$$(a' < b') \wedge (a, a'), (a, b') \in R \;\Rightarrow\; \forall c' > b', (a, c') \in R \tag{1}$$

and

$$(a < b) \wedge (a, a'), (b, a') \in R \;\Rightarrow\; \forall c > b, (c, a') \in R \tag{2}$$

Proof Since $R = R^*$, for all c', there exists c such that $(c, c') \in R$. Using (a, a'), (a, b'), $(c, c') \in R$ and Definition 4, we deduce that $(a, c') \in R$, for all $c' > b' > a'$, which proves (1). In the same way, to prove (2), we can show that $(c, a') \in R$, for all $c > b > a$ using $(a, a'), (b, a'), (c, c') \in R$ for some c'.

The conditions (1) and (2) ensure that, when the relation is mjx-closed, there is no zero preceded by two ones in a same row or a same column.

Proposition 2 *Let R be a binary relation. If R is* mjx*-closed then*

$$\begin{aligned}((a, a'), (b, b'), (c, c') \in R \;\wedge\; a > b, c \;\wedge \\ b' > a', c' \;\wedge c \neq b \wedge c' \neq a') \Rightarrow (a, b') \in R\end{aligned} \tag{3}$$

Proof If $a > b, c$, $b' > a', c'$, $c \neq b$ and $c' \neq a'$ then $(\text{mjx}(a, b, c), \text{mjx}(a', b', c')) = (a, b')$.

Proposition 3 *Let R be a binary relation such that $R{=}R^*$. Then R is* mjx*-closed if and only if it satisfies* (1)*,* (2) *and* (3)*.*

Proof ($\Rightarrow$) This is exactly Propositions 1 and 2.
($\Leftarrow$) Let R be a binary relation. If R verifies (1), (2) and (3) then, for every $(a, b') \notin R$, we show that there is no (a, a'), (b, b'), $(c, c') \in R$ that generate (a, b') by applying mjx pointwise. For the sake of contradiction, suppose that the converse is true, that is, there exist $(a, a'), (b, b'), (c, c') \in R$ such that $\text{mjx}(a, b, c) = a$, $\text{mjx}(a', b', c') = b'$ and $(a, b') \notin R$. Since $(a, a'), (b, b') \in R$ and $(a, b') \notin R$, we must have $a \neq b$ and $a' \neq b'$. Moreover, $\text{mjx}(a, b, c) = a \neq b$ implies that $c \neq b$ and therefore two possible cases: either $a > b, c$ or $a = c$. Similarly, since $\text{mjx}(a', b', c') = b' \neq a'$, we must have $c' \neq a'$ and one of the following two options: either $b' > a', c'$ or $b' = c'$. If $(a > b, c) \wedge (b' > a', c')$, then (3) implies that $(a, b') \in R$, which contradicts our hypothesis. In the case where $(a > b, c) \wedge (b' = c')$, (2) implies that $(a, b') \in R$, and this contradicts the hypothesis. If $(a = c) \wedge (b' > a', c')$ then (1) implies that $(a, b') \in R$, which also results in a contradiction. Finally, the case where $(a = c) \wedge (b' = c')$ is not possible because $(c, c') \in R$ and $(a, b') \notin R$. We conclude that R is mjx-closed.

Proposition 4 *An* mjx*-closed binary relation can be stored in $O(d)$ space.*

Proof This is a consequence of Proposition 1: instead of storing R in the form of a Boolean matrix, it suffices to store the positions of the first two 1's in each row of this matrix.

Example 4 Consider the binary relation defined on $\{0, 1, \ldots, d-1\}$ by $(|x - y| = a \ \vee \ x + y > b)$, where a and b are two constants such that $b \leq 2a$. This relation is mjx-closed. The Boolean matrix corresponding to this relation, for $d = 4$, $a = 2$ and $b = 4$, as well as the two vectors which allow its storage according to Proposition 4, are shown in Figure 1. For each row, the vectors give, respectively, the column in which the first and second 1 occurs (or ∞ if the second 1 does not exist).

	0	1	2	3
0	0	0	1	0
1	0	0	0	1
2	1	0	0	1
3	0	1	1	1

2	∞
3	∞
0	3
1	2

Fig. 1 The Boolean matrix associated with the mjx-closed relation given in Example 4, for $d = 4$. The entries of the matrix are indexed, from top to bottom and from left to right, by the elements of $\{0, 1, 2, 3\}$. A non zero entry indicates that the value pair corresponding to the entry index belongs to the relation. The two vectors used for its storage according to Proposition 4.

It is easy, but tedious, to verify that the relations given in Examples 3 and 4 satisfy the conditions (1), (2) and (3) and are, therefore, mjx-closed.

In the case where the first variable is Boolean, conditions (2) and (3) are always true, (there are not three distinct values a, b, c in this variable domain), and the only condition that needs to be checked to get the mjx closure is (1). This observation allows us to give the following example.

Example 5 The relation $(x = a) \Rightarrow (y = b \ \vee \ y \geq c)$ is mjx-closed, where x is a Boolean variable, y any variable, and a, b, c are constants.

3.2 Checking mjx closure

Checking the closure by the mjx operator can be achieved by testing the three necessary and sufficient conditions given in Proposition 3. The most simple way to do this is to verify that every triple of elements $(a, a'), (b, b'), (c, c')$ that belong to the relation does not violate these conditions. Unfortunately this method involves d^6 tests. In this section, we give an efficient solution to verify conditions (1), (2), (3) in $O(d^2)$ steps.

We assume that the relation R to be verified is given as a Boolean matrix with $R(a, b) = 1$ if and only if $(a, b) \in R$. It is reasonable to assume that computing such a Boolean matrix from a given binary relation can be performed in $O(d^2)$ steps. First, we can notice that checking conditions (1) and (2) requires $O(d^2)$ steps. Indeed, we just have to run through each row and each column only once to verify that the second 1 (if it exists) of the row or column is followed by a sequence of 1's.

In the remainder of this section, we assume that R=R^*, which can be achieved by deleting the lines and columns of zeros in R. Moreover, we assume that R satisfies conditions (1) and (2). A consequence of these assumptions is that each zero entry in R, $R(i, j)$=0, is preceded by no more than one 1 in row i and no more than one 1 in column j.

In order to verify condition (3), for all (a, b') such that $R(a, b') = 0$, we need to show that we cannot find a', b, c, c' such that $b, c < a$ and $a', c' < b'$ and

$$(R(a, a') = R(b, b') = R(c, c') = 1) \wedge (c \neq b) \wedge (c' \neq a') \tag{4}$$

To efficiently perform this test, we use the following data structures:

- $NL(i, j) = \sum_{k<j} R(i, k) =$ the number of 1's in the i^{th} row of R that are located before column j.
- $NC(i, j) = \sum_{k<i} R(k, j) =$ the number of 1's in the j^{th} column of R that are located before row i.
- $N(i, j) = \sum_{k<j} NC(i, k) =$ the total number of 1's in the sub-matrix of R composed of the rows before row i and columns before column j.
- $lig1(j) = \min\{i \mid R(i, j) = 1\} =$ the row of the first 1 in the j^{th} column of R.
- $col1(i) = \min\{j \mid R(i, j) = 1\} =$ the column of the first 1 in the i^{th} row of R.

These data structures can be initialized in $O(d^2)$ time by direct application of their respective definitions. If $R(a, b') = 0$ and since R satisfies conditions (1) and (2), we deduce that there is at most one 1 in row a located before column b' and at most one 1 in column b' located before row a. This implies that there is a single value $a' = col1(a)$ and a single value $b = lig1(b')$ such that $R(a, a') = R(b, b') = 1$ which could possibly satisfy $(b < a) \wedge (a' < b')$. If $(b < a) \wedge (a' < b')$ then, to complete the verification of condition (4), we need to check that the number of pairs (c, c') such that $R(c, c') = 1 \wedge (c \neq b) \wedge (c' \neq a') \wedge (c < a) \wedge (c' < b')$ is 0. The number of such pairs is given by the following formula, which can be calculated in constant time:

$$N(a, b') - NL(b, b') - NC(a, a') + R(b, a')$$

which gives the number of 1's in the sub-matrix of R composed of the rows before the row a and the columns before the column b' minus the number of 1's which are in row b or column a'. Note that we add the term $R(b, a')$ because it is subtracted twice from $N(a, b')$: once by subtracting $NL(b, b')$ and again by subtracting $NC(a, a')$.

We conclude, therefore, that our data structures allow the verification of condition (4) in $O(1)$ time for each pair of values (a, b'), avoiding a full search over all possible values of a', b, c and c'.

We have thus proved the following proposition.

Proposition 5 *Verifying that a binary relation is* mjx-*closed can be performed in* $O(d^2)$ *steps.*

We can even say that this complexity is optimal, since in the worst case, we need d^2 steps to read any binary relation.

3.3 Checking mjx closure for an unknown domain ordering

From a theoretical point of view, an interesting problem is to determine whether a given set of relations over a domain D is mjx-closed for some unknown ordering of D. It is already known that determining the existence of a domain ordering under which a set of relations is max-closed or median-closed is NP-complete [GC08]. Therefore it is hardly surprising that the corresponding problem for mjx-closure also turns out to be NP-complete, as we will now show. It is worth pointing out, however, that the more general class of majority polymorphisms can be detected in polynomial time [BCH$^+$13].

Proposition 6 *Given a set Γ of relations over domain D, determining whether there exists a total ordering of D such that the relations in Γ are all* mjx*-closed is NP-complete.*

Proof The problem is clearly in NP since given a certificate (a domain ordering) the set of relations can be checked for mjx-closure in polynomial time. To complete the proof, it thus suffices to provide a polynomial reduction from the well-known NP-complete problem 3-SAT. Let P_{3SAT} be an instance of 3-SAT. For each variable X_i $(i = 1, \ldots, n)$ in P_{3SAT}, we add the integers i and $i + n$ to D. We also add another element $k_0 \notin \{1, \ldots, 2n\}$ to D. We make the assignment $X_i = \texttt{true}$ $(i \in \{1, \ldots, n\})$ in P_{3SAT} if and only if $i > k_0$ in the ordering of D. We will add relations to Γ (and extra values to D) so that there exists an ordering of D under which Γ is mjx-closed if and only if the corresponding assignment to the variables X_i $(i = 1, \ldots, n)$ is a solution to P_{3SAT}. To achieve this it is sufficient to show how to code the negation $X_i = \neg X_j$ (where $j = i + n$, for each $i = 1, \ldots, n$) and how to code a ternary positive clause $X_i \vee X_j \vee X_k$ (for $i, j, k \in \{1, \ldots, 2n\}$).

	k_2	k_0	j
k_1	1	0	0
k_0	0	1	1
i	0	1	0

	k_0	i	j
k_3	0	1	1
k_4	0	1	1
k_5	0	1	1

Fig. 2 Relations to code the negation $X_i = \neg X_j$. The first line and column contain the row and column numbers. Those rows and columns that are not shown are all zeros.

To code the negation $X_i = \neg X_j$ we add the binary relations shown in Figure 2 to Γ, where $k_1, \ldots, k_5$ are domain values not occurring in other relations. From Proposition 3, we can deduce that a necessary and sufficient condition for these two relations to be mjx-closed is that the domain ordering satisfies:

$$\begin{gathered} k_1 \neq \max(k_1, k_0, i) \wedge \\ k_2 \neq \max(k_2, k_0, j) \wedge \\ ((i \neq \max(k_1, k_0, i)) \ \vee \ (j \neq \max(k_2, k_0, j))) \wedge \\ ((i > k_0) \ \vee \ (j > k_0)). \end{gathered}$$

It is easy to verify that the only possible orderings of the elements i, j, k_0 are $i > k_0 > j$ or $j > k_0 > i$. Thus, adding the two relations shown in Figure 2 to Γ imposes the constraint $X_i = \neg X_j$.

	k_6	k	k_7
i	0	0	1
k_0	0	1	0
j	1	0	0

	k_0	k_6	k_7
k_8	0	1	1
k_9	0	1	1
k_{10}	0	1	1

Fig. 3 Relations to code the clause $X_i \vee X_j \vee X_k$. The first line and column contain the row and column numbers. Those rows and columns that are not shown are all zeros.

To code the positive clause $X_i \vee X_j \vee X_k$, we add the two binary relations shown in Figure 3 to Γ, where $k_6, \ldots, k_{10}$ are domain values not occurring in other relations. A necessary and sufficient condition for these relations to be mjx-closed is that

$$\begin{aligned} & (i = \max(i, k_0, j) \quad \wedge \quad k_7 = \max(k_6, k, k_7)) \\ \vee\ & (k_0 = \max(i, k_0, j) \quad \wedge \quad k = \max(k_6, k, k_7)) \\ \vee\ & (j = \max(i, k_0, j) \quad \wedge \quad k_6 = \max(k_6, k, k_7)) \end{aligned}$$

together with $(k_6 > k_0) \vee (k_7 > k_0)$. It is tedious but easy to check that all orderings of the elements i, j, k, k_0 are possible except those in which $k_0 > i, j, k$. Thus we have $(i > k_0) \vee (j > k_0) \vee (k > k_0)$ which corresponds to the clause $X_i \vee X_j \vee X_k$.

4 Solving mjx-closed binary CSP

As outlined in Section 2, local consistency guarantees properties related to the consistency of subsets of variables or constraints. Local consistency can be enforced via problem transformations called *constraint propagation*. These transformations enable the elimination of certain inconsistent values or tuples of values. This may result in a reduction of the cost of subsequent search. There are several local consistency levels, each providing a different balance between efficient filtering and speed of search, the most well-known being arc consistency and path consistency.

4.1 Local consistency for mjx-closed binary CSP

It is known that solving a CSP instance whose relations are all closed under any majority operation, such as mjx, can be achieved either by strong 3-consistency [JCC98] or by singleton arc consistency [CDG13]. It turns out that, in the case of mjx-closed relations, an enhanced form of strong directional path consistency is sufficient (see Proposition 9).

Definition 5 Let P=(X, D, C) be a binary CSP instance.

- P is directional arc consistent if, for any pair of variables (x_i, x_j) such that $i < j$ and for any value $v_i \in D_i$, there is a value $v_j \in D_j$ such that the partial instantiation $\{(x_i, v_i), (x_j, v_j)\}$ satisfies the binary constraint between x_i and x_j, if this constraint exists.
- P is directional path consistent if, for any triple (x_i, x_j, x_k) of variables such that $i, j < k$ and for all pairs of consistent values $(v_i, v_j) \in D_i \times D_j$, there is a value v_k in D_k such that the partial instantiation $\{(x_i, v_i), (x_j, v_j), (x_k, v_k)\}$ is consistent (i.e. satisfies all constraints of C involving only x_i, x_j, x_k).

In what follows, we show that establishing a simple property related to arc consistency, that we call *weak arc consistency*, in addition to strong directional path consistency, is enough to solve mjx-closed binary CSP instances. Weak arc consistency imposes that any value belonging to the domain of a variable x_i must be part of at least one of the pairs in every relation involving x_i. Weak arc consistency amounts therefore to maintaining the following inclusions:

$$D_i \subseteq \Pi_i(R_{i,j}), \quad \text{for all} \quad (\{x_i, x_j\}, R_{i,j}) \in C \tag{5}$$

where $\Pi_i(R_{i,j}) = \{a \in D \mid (a, b) \in R_{i,j}\}$. Weak arc consistency is, indeed, weaker than arc consistency since the latter consistency level maintains the following invariant

$$D_i \subseteq \Pi_i(R_{i,j} \bowtie D_j), \quad \text{for all} \quad (\{x_i, x_j\}, R_{i,j}) \in C \tag{6}$$

where $\bowtie$ denotes the natural join with regard to the common variable x_j. Since $(R_{i,j} \bowtie D_j) \subseteq R_{i,j}$, we deduce that (6) implies (5). Moreover, it is easy to deduce from (5) that, contrary to arc consistency, weak arc consistency is preserved by domain reduction. This means that reducing the domain of any variable to one of its subsets does not affect the inclusions appearing in (5). Observe also that weak arc consistency is ensured by establishing arc consistency but is not ensured by establishing directional arc consistency.

Algorithm 1 describes a procedure, SDPC$^+$, which enforces strong directional path consistency in addition to weak arc consistency. To enforce this latter local consistency level, Algorithm 1 uses the classical projection (Π) and natural join ($\bowtie$) operations. SDPC$^+$ is a standard procedure for establishing strong directional path consistency, to which we have added steps 1 to 3 and step 10. The role of these supplementary steps is to establish weak arc consistency. First, we show that the application of SDPC$^+$ produces a strong directional path consistent instance.

Proposition 7 *Let P be a binary CSP instance, then* SDPC$^+(P)$ *is strongly directional path consistent and has the same solution set as P.*

Proof First, observe that the loop beginning at line 1 of procedure SDPC$^+$ deletes inconsistent values and does not alter the level of consistency established in the following steps. The rest of the algorithm consists of the steps needed to establish strong directional path consistency (see [Dec03], for example), to which we have added step 10. This last step deletes, from D_i, the values that cannot have any support in D_j, because these values are not part of any value pair of $R_{i,j}$. Of course, such values are inconsistent and can be removed without modifying the solution set of the instance.

Let us show that step 10 preserves the level of directional consistency established before its execution. First, notice that step 10 can only reduce value domains. We have to show that step 10 does not affect the directional arc consistency already established. At iteration k, the arcs that were already made directional arc consistent are those of the form $(x_{i'}, x_{k'})$, with $k \leq k' \leq n$ and $i' < k'$. To alter the directional arc consistency of one of these arcs, we need to reduce one of the domains $D_{k'}$ ($k \leq k' \leq n$), but, at iteration k, step 10 can only reduce the domains D_i, $1 \leq i \leq k-1$.

Moreover, step 10 cannot affect the directional path consistency that has been established at iterations k to n. Indeed, at iteration k, the paths already made consistent are $(x_{i'}, x_{j'}, x_{k'})$, with $i' < j' < k'$ and $k \leq k' \leq n$. To alter the consistency of one of these paths, it is necessary to reduce one of the domains $D_{k'}$ $(k \leq k' \leq n)$, but, at iteration k, step 10 can only reduce the domains D_i, $1 \leq i \leq k-1$.

Next, we show that SDPC$^+$ enforces weak arc consistency on binary CSP instances.

Proposition 8 *Let P be a binary* CSP *instance, then* SDPC$^+(P)$ *is weak arc consistent.*

Proof To begin with, notice that weak arc consistency of P is established by steps 1 to 3, since $D_i \cap \Pi_i(R_{i,j}) \subseteq \Pi_i(R_{i,j})$. Moreover, as we have already observed, weak arc consistency is preserved by domain reduction, that is, if the weak arc consistency is verified for a value domain D_i then it remains verified if D_i is reduced to one of its subsets. It follows that the weak arc consistency on a domain D_i may be lost only if a constraint involving x_i is updated, which can only take place at step 9. Step 10 is, therefore, applied to restore the weak arc consistency of D_i again.

Algorithm 1 Procedure SDPC$^+(X, D, C)$

1: **for all** $(\{x_i, x_j\}, R_{i,j}) \in C$ **do** - - Establishing weak arc consistency
2: $\quad D_i \leftarrow D_i \ \cap\ \Pi_i(R_{i,j})$
3: **end for**
- - Establishing directional path consistency
4: **for** $k \leftarrow |X|$ **to** 1 **do**
5: $\quad$ **for all** $i < k$ such that $(\{x_i, x_k\}, R_{i,k}) \in C$ **do**
6: $\quad\quad D_i \leftarrow D_i \ \cap\ \Pi_i(R_{i,k} \bowtie D_k)$
7: $\quad$ **end for**
8: $\quad$ **for all** $i, j < k$ such that $(\{x_i, x_k\}, R_{i,k}), (\{x_j, x_k\}, R_{j,k}) \in C$ **do**
9: $\quad\quad R_{i,j} \leftarrow R_{i,j} \ \cap\ \Pi_{i,j}(R_{i,k} \bowtie D_k \bowtie R_{j,k})$
10: $\quad\quad D_i \leftarrow D_i \ \cap\ \Pi_i(R_{i,j})$ - - Restoring weak arc consistency
11: $\quad\quad C \leftarrow C \cup \{(\{x_i, x_j\}, R_{i,j})\}$
12: $\quad$ **end for**
13: **end for**

Proposition 9 *Let P be a binary CSP instance defined over a* mjx*-closed language. Establishing strong directional path consistency together with weak arc consistency suffices to determine whether P is consistent and, if so, to calculate a solution in linear time.*

Proof Establishing strong directional path consistency does not destroy the closure under mjx, since mjx is a specific majority operation and the closure under this latter operation is preserved by enforcing strong directional path consistency [CJ06]. This is also the case for weak arc consistency. Indeed, the additional steps required by the latter consistency level affect only the domains of variables, that is, the unary relations, which remain closed under mjx.

Suppose that P is strongly directional path consistent according to the increasing order of variable index and also weak arc consistent. If P contains an empty domain then no solution exists. Otherwise, let $(a_1, \ldots, a_{k-1})$ be a partial solution, in other words a value assignment to variables $(x_1, \ldots, x_{k-1})$ which satisfies all the constraints involving these variables. By directional arc consistency, we can find such a partial solution for $k = 3$. We show that it is always possible (for $k = 4, \ldots, n$) to extend $(a_1, \ldots, a_{k-1})$ to a partial solution $(a_1, \ldots, a_k)$. It will follow, by induction, that a solution can be found in linear time.

Let $R_{i,j}$ be the relation defining the constraint on (x_i, x_j) and $R_{i,j}(a_i)$ the set of values $b \in D_j$ such that $(a_i, b) \in R_{i,j}$. Thanks to directional arc consistency, which follows from strong directional path consistency, we must have $R_{i,k}(a_i) \neq \emptyset$, for all $i,\ 1 \leq i \leq k-1$. We therefore distinguish two cases:

1. $\exists i,\ 1 \leq i \leq k-1$ such that $R_{i,k}(a_i)$ is a singleton. Let a_k be the only element of $R_{i,k}(a_i)$. By directional path consistency, all $a_i,\ 1 \leq i \leq k-1$ are necessarily consistent with a_k. It follows that $(a_1, \ldots, a_{k-1}, a_k)$ is a partial solution.
2. All $R_{i,k}(a_i),\ 1 \leq i \leq k-1$ contain at least two elements. Let us designate by m_k the maximum element of D_k. We will prove that m_k is consistent with all $a_i,\ 1 \leq i \leq k-1$. Since P is weak arc consistent, there must exist a'_i such that $(a'_i, m_k) \in R_{i,k}$, otherwise we would have $D_k \nsubseteq \Pi_k(R_{i,k})$. Denote by b_k and b'_k any two distinct elements of $R_{i,k}(a_i)$. Recall that the existence of such elements is guaranteed by $|R_{i,k}(a_i)| \geq 2$. We conclude that $(a_i, b_k), (a_i, b'_k), (a'_i, m_k) \in R_{i,k}$, therefore $(\mathrm{mjx}(a_i, a_i, a'_i), \mathrm{mjx}(b_i, b'_i, m_k)) = (a_i, m_k) \in R_{i,j}$. We conclude that $(a_1, \ldots, a_{k-1}, m_k)$ is a partial solution.

In fact, procedure SDPC$^+$ solves a much wider class than the one containing all mjx-closed binary CSP instances. Indeed, SDPC$^+$ may be used to solve CSP(Γ) where Γ is closed under any (possibly unknown) majority polymorphism f, as we now show.

Proposition 10 *Let P be a binary CSP instance defined over a language Γ which is f-closed where f is a majority polymorphism. Establishing strong directional path consistency together with weak arc consistency suffices to determine whether P is consistent and, if so, calculate a solution in time $O(dm)$, where the number of constraints m is assumed to be at least n.*

Proof We assume that P is weak arc consistent and strong directional path consistent according to the variable order $x_1, \ldots, x_n$. By directional arc consistency, there is a partial solution (a_1, a_2) to P on variables (x_1, x_2). We use an inductive proof to show that (a_1, a_2) can be extended to a complete solution. Let $(a_1, \ldots, a_{k-1})$ be a partial solution on variables $(x_1, \ldots, x_{k-1})$, where $3 \leq k \leq n$. To complete the inductive proof it suffices to show that this implies that $\exists a_k \in D_k$ such that $(a_1, \ldots, a_k)$ is a partial solution on variables $(x_1, \ldots, x_k)$. This will imply the existence of a backtrack-free algorithm of total complexity $O(dm)$ to find a complete solution, which simply exhausts over all possible assignments to the next variable x_k, checking constraints as it goes.

By directional path consistency, $\forall i, j$ such that $1 \leq i < j \leq k-1$, $\exists c_k^{ij} \in D_k$ such that

$$(a_i, c_k^{ij}) \in R_{i,k} \quad \wedge \quad (a_j, c_k^{ij}) \in R_{j,k} \tag{7}$$

We will show by another induction that $\forall j = 2, \ldots, k-1$, the following hypothesis H(j) is true.

$$\mathrm{H}(j): \qquad \exists d_k^j \in D_k \ \text{ such that } \ \forall i = 1, \ldots, j,\ (a_i, d_k^j) \in R_{i,k}.$$

Let $d_k^2 = c_k^{12}$. Then, by (7), H(2) holds. To complete the proof it suffices to show that H$(k-1)$ holds, since in this case we can set $a_k = d_k^{k-1}$ to obtain a partial solution $(a_1, \ldots, a_k)$.

So suppose that H$(j-1)$, where $3 \leq j \leq k-1$: we will show that this implies H(j). To do so, we use yet another induction. We will show by induction on h that $\forall h = 1, \ldots, j-1$, the following hypothesis H$'(h)$ holds.

$$\mathrm{H}'(h): \quad \exists b_k^h \in D_k \text{ such that } \quad (\forall i = 1, \ldots, h,\ (a_i, b_k^h) \in R_{i,k}) \ \wedge\ (a_j, b_k^h) \in R_{j,k}$$

Let $b_k^1 = c_k^{1j}$. Then, by (7), H$'(1)$ holds. If H$'(j-1)$ holds, then by setting $d_k^j = b_k^{j-1}$ we will have that H(j) holds which will then complete the proof.

So suppose that H$'(h-1)$ holds, where $2 \leq h \leq j-1$: we will show that this implies H$'(h)$. Let

$$b_k^h \;=\; f(d_k^{j-1}, b_k^{h-1}, c_k^{hj}).$$

By the hypothesis H$(j-1)$, and since $h \leq j-1$, we have

$$\forall i = 1, \ldots, h,\ (a_i, d_k^{j-1}) \in R_{i,k}. \tag{8}$$

By the hypothesis H$'(h-1)$, we have

$$(\forall i = 1, \ldots, h-1,\ (a_i, b_k^{h-1}) \in R_{i,k}) \quad \wedge \quad (a_j, b_k^{h-1}) \in R_{j,k}. \tag{9}$$

And, by (7), we have

$$(a_h, c_k^{hj}) \in R_{h,k} \quad \wedge \quad (a_j, c_k^{hj}) \in R_{j,k}. \tag{10}$$

Now, by weak arc consistency,

$$\forall i = 1, \ldots, h-1,\ \exists a_i' \in D_i \text{ such that } (a_i', c_k^{hj}) \in R_{i,k} \tag{11}$$

$$\exists a_h' \in D_h \text{ such that } (a_h', b_k^{h-1}) \in R_{h,k} \tag{12}$$

$$\exists a_j' \in D_j \text{ such that } (a_j', d_k^{j-1}) \in R_{j,k} \tag{13}$$

Thus, by closure of these relations under the majority operation f, we can deduce from (8), (9) and (11) that

$$\forall i = 1, \ldots, h-1,\ (f(a_i, a_i, a_i'), f(d_k^{j-1}, b_k^{h-1}, c_k^{hj})) \;=\; (a_i, b_k^h) \in R_{i,k} \tag{14}$$

Similarly, from (8), (12) and (10) we can deduce that

$$(f(a_h, a_h', a_h), f(d_k^{j-1}, b_k^{h-1}, c_k^{hj})) \;=\; (a_h, b_k^h) \in R_{h,k} \tag{15}$$

And finally, from (13), (9) and (10) we can deduce that

$$(f(a_j', a_j, a_j), f(d_k^{j-1}, b_k^{h-1}, c_k^{hj})) \;=\; (a_j, b_k^h) \in R_{j,k} \tag{16}$$

Bringing together (14), (15) and (16), we have that H$'(h)$ holds, which completes the proof.

Returning to the specific case of the polymorphism mjx, procedure SDPC$^+$ shows that efficient computation of intersection $R \cap S$ and composition $\Pi_{x,y}$ $(R \bowtie S)$ of relations is critical for the efficiency of solving mjx-closed CSPs. In the two following subsections we show that these two operations can be achieved in $O(d)$ time if relations R and S are closed under mjx, instead of $O(d^2)$ and $O(d^3)$ for arbitrary relations.

4.2 Efficient implementation

In what follows, we present an efficient implementation of each of the components of a solution algorithm dedicated to solving mjx-closed binary CSPs.

4.2.1 Intersection of two mjx*-closed relations*

The intersection operation is defined on pairs of relations as follows:

$$R \cap S = \{(u, v) \in D^2 \mid (u, v) \in R \ \wedge (u, v) \in S\}$$

Hereafter, we will use the function $min\_2(E)$ which returns the second smallest value of the set E. We assume that each relation R is stored, in accordance with Proposition 4, by means of the following variables:

- $col1_R(i) \;=\; \min\{j \mid R(i, j) = 1\}$, that is, the column of the first 1 in the i^{th} row of R;
- $col2_R(i) \;=\; \min\_2\{j \mid R(i, j) = 1\}$, that is, the column of the second 1 in the i^{th} row of R.

In the case where the first or the second 1 does not exist, the associated variable $col1_R(i)$ or $col2_R(i)$ will take a default value greater that d, which we denote by ∞. Relying on the fact that the second 1 in a row is followed by a sequence of 1's, the intersection $T = R \cap S$ can be computed according to the following rules: $\forall i$,

$$col1_T(i) \;=\; \begin{cases} col1_R(i) & \text{if } col1_R(i) = col1_S(i) \\ col1_R(i) & \text{if } col1_R(i) \geq col2_S(i) \\ col1_S(i) & \text{if } col1_S(i) \geq col2_R(i) \\ \max(col2_R(i), col2_S(i)) & \text{otherwise} \end{cases}$$

and

$$col2_T(i) \;=\; \begin{cases} \max(col2_R(i), col2_S(i)) & \text{if } col1_R(i) = col1_S(i) \\ col2_S(i) & \text{if } col1_S(i) \geq col2_R(i) \\ col2_R(i) & \text{if } col1_R(i) \geq col2_S(i) \\ \max(col2_R(i), col2_S(i)) + 1 & \text{otherwise} \end{cases}$$

For simplicity, we assume here that $D_i = \{1, \ldots, d\}$ and $d+1 = \infty$. For each value of i, these calculations are performed in constant time. So the calculation of $R \cap S$ is performed in $O(d)$ time. Since we must return the $2d$ values $col1_T(i)$, $col2_T(i)$ which represent the relation T, we can deduce that this complexity is optimal.

4.2.2 Composition of two mjx*-closed relations*

In this section, we focus on the second operation that intervenes in enforcing directional path consistency, namely, relation composition. We limit ourselves to the composition of binary relations, because any unary relation may be easily expressed by a binary one, as it will be shown latter. First, let us express binary relation composition via the more classical natural join and projection operations. Let $R_{i,k}$ and $R_{k,j}$ be two binary relations sharing attribute k, then the composition of $R_{i,k}$ and $R_{k,j}$, denoted by $R_{i,k} \circ R_{k,j}$, can be expressed as follows:

$$R_{i,k} \circ R_{k,j} \;=\; \Pi_{i,j}(R_{i,k} \bowtie R_{k,j})$$

where the natural join is performed with regard to the common attribute k.

The composition of two binary relations R and S can also be defined in extension, by means of a more explicit formula, as follows:

$$R \circ S = \{(u, v) \in D^2 \mid \exists w \in D, (u, w) \in R \wedge (w, v) \in S\}$$

It is well established that polymorphisms are preserved under the projection and the joint operations [CJ06]. Thus, if R and S are mjx-closed then so is $T = R \circ S$. It follows that we can represent T using the values $col1_T(i)$, $col2_T(i)$ (for each row i in the matrix that represent T). For simplicity of presentation, we assume that matrices R and S are both of size $d \times d$. To optimize the calculations, we will use the following data structures:

- $m1_R(i) = \min\{j \mid \exists k \geq i, R(k, j) = 1\}$, that is, the first column j such that there is a 1 in at least one of the rows $k \geq i$
- $m2_R(i) = \min\_2\{j \mid \exists k \geq i, R(k, j) = 1\}$, that is, the second column j such that there is a 1 in at least one of the rows $k \geq i$.

In the composition algorithm given below, the first loop incrementally computes the values of $m1_S(i)$ and $m2_S(i)$ for each row i, beginning with the last row. In order to calculate row i in the matrix $T = R \circ S$, we distinguish three cases, according to the form of row i in R (1) $(0, \ldots, 0)$, (2) $(0, \ldots, 0, 1, 0, \ldots, 0)$ or (3) $(0, \ldots, 0, 1, 0, \ldots, 0, 1, \ldots, 1)$. In each case, we can deduce the form of row i in T directly from the definition of the composition operation.

1. If row i of R is all zero then the row i of T will also be all zero.
2. If row i of R includes only one 1 (in column j) then row i of T will be identical to row j of S.
3. If row i of R contains at least two 1's (with the first two 1's in columns j and $k > j$), then $col1_T(i) = \min\{col1_S(j), m1_S(k)\}$ and $col2_T(i) = \min\_2\{col1_S(j)$, $col2_S(j)$, $m1_S(k)$, $m2_S(k)\}$.

The input parameters of Algorithm 2 are two relations R and S, which are assumed to be mjx-closed and stored in the form of the positions of the first two 1's in each row ($col1_R$ and $col2_R$ for R and $col1_S$ and $col2_S$ for S). The first operation of the algorithm is to fill data structures $m1_S$ and $m2_S$, described above (first loop **for**). The algorithm then calculates the composition according to the three rules given above (second loop **for**). The result of the algorithm, for inputs R and S, are two vectors which store, respectively, the positions of the first and second 1 in each row of the matrix associated with the relation $R \circ S$. The complexity of our composition algorithm is $O(d)$. Since we must return the $2d$ values $col1_T(i)$, $col2_T(i)$, we can deduce that this complexity is optimal.

Algorithm 2 Function $Composition(R, S)$: $T = R \circ S$

- - R and S are given in the form of vectors $col1_R$, $col2_R$ and $col1_S$,$col2_S$
- - The result, $T = R \circ S$, is calculated in $col1_T$ and $col2_T$

$m_1 \leftarrow \infty$
$m_2 \leftarrow \infty$
for $i = d$ to 1 **do**
  $m1_S(i) \leftarrow min(col1_S(i), m_1)$
  $m2_S(i) \leftarrow min\_2(col1_S(i), col2_S(i), m_1, m_2)$
  $m_1 \leftarrow m1_S(i)$
  $m_2 \leftarrow m2_S(i)$
end for
for $i = 1$ to d **do**
  j $\leftarrow col1_R(i)$
  k $\leftarrow col2_R(i)$
  if $(k = \infty)$ **then**
    if $(j = \infty)$ **then** - - case 1: line i of R contains only zeros.
      $col1_T(i) \leftarrow \infty$
      $col2_T(i) \leftarrow \infty$
    else - - case 2: line i of R contains one 1.
      $col1_T(i) \leftarrow col1_S(j)$
      $col2_T(i) \leftarrow col2_S(j)$
    end if
  else - - case 3: line i of R contains two 1's.
    $E \leftarrow \{col1_S(j),\ col2_S(j),\ m1_S(k),\ m2_S(k)\}$
    $col1_T(i) \leftarrow \min(E)$
    $col2_T(i) \leftarrow \min\_2(E)$
  end if
end for
return $(col1_T,\ col2_T)$

In the next section we present in detail the algorithm SDPC$^+$ to establish strong directional path consistency and weak arc consistency, designed for instances in which all relations are mjx-closed. It has $O(n^3d)$ time complexity and $O(n^2d)$ space complexity.

4.2.3 A solution algorithm for mjx*-closed CSP*

In this section, we describe an algorithm that establishes the level of local consistency needed to solve binary CSP instances defined by means of mjx-closed relations. Recall that, according to Proposition 9, it suffices to establish strong directional path consistency enhanced with weak arc consistency.

Algorithm 3 is a version of procedure SDPC$^+$ which takes into account the specific features of mjx-closed relations. It begins by establishing weak arc consistency via the calls to procedures *WeakAC* and *WeakACInv*, both of these procedures having a linear complexity $O(d)$ (see Algorithms 4 and 5). Following the update of relation $R_{i,j}$, (see line 12), we need to restore the weak arc consistency for the arcs (x_i, x_j) such that $i, j < k$ (see the calls to procedures *WeakAC* and *WeakACInv* at lines 13 and 14).

The intersection and composition functions described above are called in order to achieve these two basic operations in linear time. To compute the term $\Pi_{i,j}(R_{i,k} \bowtie D_k \bowtie R_{j,k})$, (see line 10 of Algorithm 1), by means of function *Composition*, the domain D_k is transformed into a binary relation denoted by $R_{k,k}$ and defined as follows:

$$R_{k,k}(a,b) \;=\; \begin{cases} 1 \text{ if } a = b \;\wedge\; a \in D_k \\ 0 \text{ otherwise} \end{cases}$$

It is easy to see that $R_{k,k}$ is mjx-closed. So, it can be stored in two vectors $col1_{R_{k,k}}$ and $col2_{R_{k,k}}$. These vectors are initialized by function *RelBin*, which executes in linear time. This allows us to compute the term $\Pi_{i,j}(R_{i,k} \bowtie D_k \bowtie R_{j,k})$ by means of two calls to function *Composition*, since $\Pi_{i,j}(R_{i,k} \bowtie D_k \bowtie R_{j,k}) = R_{i,k} \circ R_{k,k} \circ R_{k,j}$.

Finally, since all the steps performed by Algorithm 3, inside its **for** loops, are linear in d, we deduce that the overall time complexity of this algorithm is $O(n^3 d)$. The space complexity is, in turn, mainly due to storage space required for the relations. In a binary CSP, we may have $O(n^2)$ constraints and since all these constraints are mjx-closed relations, and then can be stored in $O(d)$ space, we obtain a space complexity of $O(n^2 d)$.

Algorithm 3 Procedure MJX-SDPC$^+(X, D, C)$

for all $i < j$ such that $(\{x_i, x_j\}, R_{i,j}) \in C$ **do**
    WeakAC$(i, j, D, R_{i,j})$
    WeakACInv$(i, j, D, R_{i,j})$
end for
for all $k \leftarrow |X|$ **to** 1 **do**
    for all $i < k$ such that $(\{x_i, x_k\}, R_{i,k}) \in C$ **do**
        ReviseDomain$(i, k, D, R_{i,j})$
    end for
    $R_{k,k} \leftarrow$ RelBin(D_k)
    for all $i < j < k$ such that $(\{x_i, x_k\}, R_{i,k}), (\{x_j, x_k\}, R_{j,k}) \in C$ **do**
        $T \leftarrow$ Composition(Composition$(R_{i,k}, R_{k,k}), R_{j,k})$
        $R_{i,j} \leftarrow$ Intersection$(R_{i,j}, T)$
        WeakAC$(i, j, D, R_{i,j})$
        WeakACInv$(i, j, D, R_{i,j})$
        $C \leftarrow C \cup \{(\{x_i, x_j\}, R_{i,j})\}$
    end for
end for

Algorithm 4 Procedure WeakAC$(i, j, D, R_{i,j})$

for all $a \in D_i$ **do**
    if $col1 R_{i,j}(a) = \infty$ **then**
        $D_i \leftarrow D_i \setminus \{a\}$
    end if
end for

5 Conclusion

In this article, we presented a new polynomial relational class, namely, the class of binary CSP instances in which all relations are closed by the mjx operator. This ternary majority operator is a function that returns the maximum of its

Algorithm 5 Procedure WeakACInv($i, j, D, R_{i,j}$)

```
for all b ∈ D_j do
    T[b] ← 0
end for
minCol2 ← ∞
for all a ← 1 to d do
    T[col1R_{i,j}(a)] ← 1
    minCol2 ← min(minCol2, col2R_{i,j}(a))
end for
for all b ∈ D_j do
    if T[b] = 0 ∧ b < minCol2 then
        D_j ← D_j \ {b}
    end if
end for
```

Algorithm 6 Procedure ReviseDomain($i, j, D, R_{i,j}$)

```
m_j ← max(D_j)
for all a ∈ D_i do
    if col1R_{i,j}(a) ∉ D_j ∧ col2R_{i,j}(a) > m_j then
        D_i ← D_i \ {a} - - a has no support in D_j
    end if
end for
```

arguments when they are all distinct. We have shown, with examples, that some useful relations are mjx-closed. We also provided an alternative characterization of mjx-closed relations, which allowed us to demonstrate that such relations can be stored in $O(d)$ space. Another contribution is an optimal $O(d^2)$ algorithm which identifies mjx-closed relations for a fixed domain ordering.

Even if only few instances encountered in practice fall into our tractable class, our algorithms could be used as an efficient filtering technique on a global constraint consisting of all mjx-closed constraints in an instance. The set of these constraints can be efficiently identified in $O(md^2)$ time and hence they do not need to be identified as mjx-closed by the user.

The use of appropriate data structures for storing mjx-closed relations provides more than a mere gain in memory, since it has allowed us to propose two new algorithms, with linear complexity ($O(d)$), to calculate the intersection and composition of two mjx-closed relations. As a consequence, we optimized the time complexity of strong directional path consistency which falls to $O(n^3d)$ for mjx-closed relations. Next, we showed that strong directional path consistency enhanced by a new local consistency level, called weak arc consistency, ensure the consistency of binary CSPs with mjx-closed relations, resulting in a $O(n^3d)$ time complexity and a $O(n^2d)$ space complexity.

These results show that the study of a polynomial relational class defined by a judicious choice of polymorphism may be interesting in terms of a compact storage method and an efficient solution algorithm. It would be interesting in the future to study other relational classes defined by simple polymorphisms to find other similar results or even to try to establish a theory about relational classes that can be solved in low-order polynomial time. The fact that relational classes closed under any majority polymorphism can also be solved by strong directional path consistency and weak arc consistency, as we prove in this paper, may provide a first step in this direction.

References

[BCH+13] Christian Bessiere, Clément Carbonnel, Emmanuel Hebrard, George Katsirelos, and Toby Walsh. Detecting and exploiting subproblem tractability. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. IJCAI/AAAI, 2013.

[BJK05] Andrei A. Bulatov, Peter G. Jeavons, and Andrei A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34:720–742, 2005.

[BK14] Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61(1):3, 2014.

[BKW12] Libor Barto, Marcin Kozik, and Ross Willard. Near unanimity constraints have bounded pathwidth duality. In *LICS*, pages 125–134. IEEE, 2012.

[BRYZ05] Christian Bessière, Jean-Charles Régin, Roland H. C. Yap, and Yuanlin Zhang. An optimal coarse-grained arc consistency algorithm. *Artif. Intell.*, 165(2):165–185, 2005.

[CC16] Clément Carbonnel and Martin C. Cooper. Tractability in constraint satisfaction problems: a survey. *Constraints*, 21(2):115–144, 2016.

[CDG13] Hubie Chen, Víctor Dalmau, and Berit Grußien. Arc consistency and friends. *J. Log. Comput.*, 23(1):87–108, 2013.

[CJ06] David A. Cohen and Peter G. Jeavons. The complexity of constraint languages. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, pages 245–280. Elsevier, 2006.

[CJS10] Martin C. Cooper, Peter G. Jeavons, and András Z. Salamon. Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. *Artif. Intell.*, 174(9-10):570–584, 2010.

[CMTZ14] Martin C. Cooper, Achref El Mouelhi, Cyril Terrioux, and Bruno Zanuttini. On broken triangles. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 9–24. Springer, 2014.

[Coo89] Martin C. Cooper. An optimal k-consistency algorithm. *Artif. Intell.*, 41(1):89–95, 1989.

[DBH99] Yves Deville, Olivier Barette, and Pascal Van Hentenryck. Constraint satisfaction over connected row convex constraints. *Artif. Intell.*, 109(1-2):243–271, 1999.

[Dec03] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[DP87] Rina Dechter and Judea Pearl. Network-based heuristics for constraint-satisfaction problems. *Artif. Intell.*, 34(1):1–38, 1987.

[Fre85] Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM*, 32(4):755–761, October 1985.

[FV98] Tomás Feder and Moshe 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, 1998.

[GC08] Martin J. Green and David A. Cohen. Domain permutation reduction for constraint satisfaction problems. *Artif. Intell.*, 172(8-9):1094–1118, 2008.

[GLS00] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.

[Gro07] Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1–24, March 2007.

[JC95] Peter Jeavons and Martin Cooper. Tractable constraints on ordered domains. *Artif. Intell.*, 79(2):327–339, 1995.

[JCC98] Peter Jeavons, David A. Cohen, and Martin C. Cooper. Constraints, consistency and closure. *Artif. Intell.*, 101(1-2):251–265, 1998.

[Jea98] Peter G. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.

[Naa13] Wady Naanaa. Unifying and extending hybrid tractable classes of CSPs. *J. Exp. Theor. Artif. Intell.*, 25(4):407–424, 2013.

[vHK06] Willem-Jan van Hoeve and Irit Katriel. Global constraints. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 169–208. Elsevier, 2006.