A Hybrid Tractable Class for Non-Binary CSPs*

Achref El Mouelhi **Philippe Jégou** **Cyril Terrioux**
Aix Marseille Université, CNRS, ENSAM, Université de Toulon,
LSIS UMR 7296, 13397 Marseille, France
{achref.elmouelhi,philippe.jegou,cyril.terrioux}@lsis.org

Abstract

Find new islands of tractability, that is classes of CSP instances for which polytime algorithms exist, is a fundamental task in the study of constraint satisfaction problems. The concept of hybrid tractable class, which allows to deal simultaneously with the restrictions of languages and, for example, the satisfaction of structural properties, is an approach which has already shown its interest in this domain. Here we study a hybrid class for non-binary CSP instances. With this aim in view, we consider the Broken Triangle Property (BTP) introduced in [8]. While this tractable class has been defined for binary instances, the authors have suggested to extend it to instances with constraints of arbitrary arities, using the dual representation of such CSPs. We develop this idea by proposing a new definition without exploiting the dual representation, but using a semantic property associated to the compatibility relations of the constraints. This class is called DBTP for Dual Broken Triangle Property. We study it in depth, firstly to show that it is tractable. Then we compare it to some known classes. In particular, we prove that DBTP is incomparable with BTP and that it includes some well known tractable classes of CSPs such as β-acyclic CSPs. Then, we compare it with the Hyper-k-Consistency, which allows us to also present new results for BTP. Finally, we analyse DBTP from a practical viewpoint, by first highlighting that some benchmarks which are classically used to compare the solvers are included in DBTP and then by explaining the efficiency of solvers of the state of the art on such instances thanks to their membership of the DBTP class.

1 Introduction

Constraint Satisfaction Problems (CSPs, see [25] for a state of the art) provide an efficient way of formulating problems in computer science, especially in Artificial Intelligence. Numerous and various problems can be modelled as CSPs like, for instance, graph coloring, frequency assignment problem, scheduling problem or Boolean satisfiability problem (SAT). The CSP problem can be considered as the problem of checking whether a finite set X of variables can be assigned in their domains of values given by D, while satisfying simultaneously a set C of constraints.

*This paper is an extension of [11]. The final publication is available at Springer via http://dx.doi.org/10.1007/s10601-015-9185-y

Formally, a *CSP instance* $P = (X, D, C)$ is defined by a set X of n variables (denoted $x_1, ..., x_n$), a set of domains $D = \{d_1, ..., d_n\}$ (d_i is the set of the possible values for the variable x_i) and a set C of e constraints (denoted $c_1, ..., c_e$). Each constraint c_i involves a set of variables called the *scope* of c_i and denoted $S(c_i)$. A constraint c_i allows a set of tuples over $\prod_{x_j \in S(c_i)} d_j$ defined by the relation $R(c_i)$ (i.e. we assume here that the relations are represented by sets of allowed tuples). $r_i = |S(c_i)|$ is the *arity* of the constraint c_i while r denotes the largest arity and $\rho = max_{c_i \in C}\{|R(c_i)|\}$ the size of the largest relation. In the following, without loss of generality, we assume that the CSP are normalized (i.e. $\forall c_i, c_j \in C, c_i \neq c_j, S(c_i) \neq S(c_j)$ [3]). Usually, we distinguish binary constraints whose arity is equal to 2 from non-binary ones. Likewise, binary CSPs (CSPs for which all the constraints are binary) are considered separately from CSPs with constraints of arbitrary arities. For binary CSPs, we will denote by c_{ij} the constraint involving x_i and x_j. For both binary CSPs and CSPs of arbitrary arity, the problem of deciding whether a solution (i.e. an assignment of a value to each variable which satisfies all the constraints) exists is NP-complete.

Although the problem CSP is NP-complete, there exist classes of instances that can be solved (and often recognized) in polynomial time. These classes are called "tractable classes" and rely on some properties that can be verified by the instances. There are two main kinds of such properties. The first one concerns the properties of the structure of the CSP instance which is represented by a hypergraph (a graph in the binary case), called the *constraint (hyper)graph*, whose vertices correspond to variables and edges to the constraint scopes. For example, it is well known that solving a tree-structured binary CSP (i.e. deciding whether it has a solution) can be achieved in linear time [14]. Another kind of properties is related to restrictions on the language defining the constraints. These restrictions concern the domains and/or the compatibility relations associated with the constraints. For example, it is the case for the class of "0-1-all constraints" (ZOA [7]). More recently, some tractable classes have been proposed which are related to these two kinds of properties, such as the Broken Triangle Property (BTP [8]). Their interest is that they are able to take into account both language and structure restrictions. They are thus sometimes called "hybrid classes".

In this paper, we study a hybrid tractable class called *DBTP* for *Dual Broken Triangle Property*. So, this class is based on the concept of "Broken Triangle" which is the basis of BTP. While BTP is only defined for binary constraints, DBTP is defined for CSPs whose constraints have arbitrary arities. Using the dual representation of CSPs, we can consider that this class has been firstly (and briefly) proposed in [8], as a non-binary version of BTP. However, we can also define DBTP by a semantic property related to the compatibility of tuples appearing in triples of relations associated to constraints, without an explicit link to the dual representation. But we show that these two definitions are equivalent (see Theorem 3). Nevertheless, DBTP is a tractable class quite different from BTP. For example, we prove that DBTP is not a generalization of BTP to constraints of arbitrary arity since in the case of binary CSPs, BTP and DBTP are formally different (see Theorem 12). Another example of these differences is related to the fact that DBTP is a conservative property for the filtering of domains and for the filtering of relations while BTP is conservative only for the filtering of domains. Moreover, we show that this tractable class includes simultaneously, structural classes such as *β-acyclic* CSPs but also classes defined by language restrictions. We also establish that DBTP is incomparable with many well known tractable classes (e.g. ZOA [7],

row-convex [1] or max-closed [19]).

As mentioned above, we prove that DBTP is a conservative property for many classical filterings like arc-consistency or pairwise consistency. It ensues that DBTP seems to have a real practical interest since any instance satisfying DBTP can be solved in polytime using algorithms similar to MAC (Maintaining Arc-Consistency [26]) or RFL (Real Full Look-ahead [23]).

This paper is organized as follows. In Section 2, we introduce the class DBTP and provide its main features. Then in Section 3, we study the relationship between BTP and DBTP and show that DBTP includes β-acyclic CSPs. Section 4 examines the relationship between DBTP and other well known tractable classes. Then, due to the relationship between BTP and hyper-3-consistency, we study the links between (D)BTP and (hyper-)k-consistency. Finally, we study DBTP from a practical viewpoint by showing that some benchmarks of the CSP 2008 Competition have the DBTP property and by making the links with their solving efficiency before concluding and giving some perspectives in Section 7.

2 The Tractable Class DBTP

In this section, we first define the DBTP class and present some of its properties (notably its tractability) before studying the effects of filtering on DBTP instances and their consequences on the efficiency of their solving by solvers of the state of the art.

2.1 Definition and Properties

First, we recall the BTP property on which the DBTP property relies:

Definition 1 (Broken Triangle Property [8]) *A CSP instance (X, D, C) satisfies the **Broken Triangle Property** (BTP) w.r.t. the variable ordering $<$ if, for all triples of variables (x_i, x_j, x_k) s.t. $x_i < x_j < x_k$, s.t. $(v_i, v_j) \in R(c_{ij})$, $(v_i, v_k) \in R(c_{ik})$ and $(v_j, v'_k) \in R(c_{jk})$, then either $(v_i, v'_k) \in R(c_{ik})$ or $(v_j, v_k) \in R(c_{jk})$. If neither of these two tuples exist, $\langle (v_i, v_j), (v_i, v_k), (v_j, v'_k) \rangle$ is called a **Broken Triangle on** x_k.*
Let BTP be the set of the instances for which BTP holds w.r.t. some variable ordering.

The BTP property is relative to the compatibility between the values of domains which can be graphically visualized on the micro-structure graph.

Definition 2 (Micro-structure [21]) *The **micro-structure** of a binary CSP instance $P = (X, D, C)$ is the undirected graph $\mu(P) = (V, E)$ where $V = \{(x_i, v_i) : x_i \in X, v_i \in d_i\}$ and $E = \{ \{(x_i, v_i), (x_j, v_j)\} : i \neq j, c_{ij} \notin C$ or $(v_i, v_j) \in R(c_{ij})\}$.*

As each of these compatibilities involves as many values as the arity of the considered constraint, such a property cannot be easily generalized to non-binary CSPs. So a natural alternative[1] consists in considering the compatibilities between the relations through the notion of dual of a CSP instance. Before defining formally the notion of dual, we introduce the notation $t[Y']$, which represents the restriction of the tuple t of $\prod_{x_i \in Y} d_i$ (where Y is a subset of variables) to the variables of the subset $Y' \subseteq Y$.

[1]Such an idea has already been introduced in [8] but it was just mentioned briefly and thus, it was not studied in depth.

Definition 3 (Dual) *The **dual** of the CSP instance $P = (X, D, C)$ is the binary CSP instance $P^d = (X^d, D^d, C^d)$ where each constraint c_i of C is associated to the variable x_i^d of X^d whose domain d_i^d is defined by the tuples t_i of $R(c_i)$ s.t. $\forall x_j \in S(c_i), t_i[\{x_j\}] \in d_j$, and a constraint c_{ij}^d of C^d links two variables x_i^d and x_j^d of X^d if the corresponding constraints c_i and c_j of C share at least a variable (i.e. $S(c_i) \cap S(c_j) \neq \emptyset$). The relation $R(c^d)$ is defined by the tuples $(t_i, t_j) \in d_i^d \times d_j^d$ s.t. $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$.*

It is well known that, for any CSP P, P has a solution iff P^d has a solution.

We now define the DBTP property:

Definition 4 (Dual Broken-Triangle Property) *A CSP instance P satisfies the **Dual Broken Triangle Property** (DBTP) w.r.t. the constraint ordering $\prec$ iff the dual of P satisfies BTP w.r.t. $\prec$.*
Let $DBTP$ be the set of the instances for which the DBTP property holds for some constraint ordering.

We can observe graphically the DBTP property on the micro-structure of the dual of the original instance. For instance, Figure 1(a) represents the micro-structure of the dual instance of a CSP P with three constraints c_1, c_2 and c_3. In this example, we consider four tuples, $t_1 \in R(c_1)$, $t_2 \in R(c_2)$ and $t_3, t'_3 \in R(c_3)$ s.t. $t_1[S(c_1) \cap S(c_2)] = t_2[S(c_1) \cap S(c_2)]$, $t_1[S(c_1) \cap S(c_3)] = t_3[S(c_1) \cap S(c_3)]$, $t_2[S(c_2) \cap S(c_3)] = t'_3[S(c_2) \cap S(c_3)]$, $t_1[S(c_1) \cap S(c_3)] \neq t'_3[S(c_1) \cap S(c_3)]$ and $t_2[S(c_2) \cap S(c_3)] \neq t_3[S(c_2) \cap S(c_3)]$. If we consider the ordering $c_1 \prec c_2 \prec c_3$, P does not satisfy DBTP w.r.t. $\prec$. Now, if we have P' (see Figure 1(b)) s.t. either t_1 and t'_3 (dotted edge), or t_2 and t_3 (dashed edge) or both are compatible, then P' satisfies DBTP according to $\prec$.

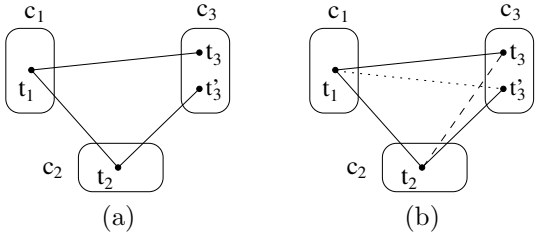


Figure 1: Illustration of DBTP on the constraints c_1, c_2 and c_3.

The class $DBTP$ differs necessarily from the class BTP since $DBTP$ may contain non-binary instances while BTP is restricted to binary instances. It follows a natural question about the comparison of these two classes in the particular case of binary CSPs. In particular, a binary instance may satisfy DBTP while not satisfying BTP. For instance, Figure 2(b) depicts the micro-structure of a binary instance while Figure 2(a) represents the micro-structure of its dual. This instance is DBTP w.r.t. the ordering $c_{ij} \prec c_{jk} \prec c_{ik}$ but not BTP since $\langle (c, e), (c, a), (e, b) \rangle$, $\langle (b, e), (b, d), (e, c) \rangle$ and $\langle (b, d), (b, e), (d, f) \rangle$ are broken triangles respectively on x_i, x_j and x_k. Note that, given three variables (respectively constraints), the existence of a broken triangle on each of them with respect to the two other is a sufficient condition in order to prevent the existence of a suitable ordering w.r.t. to BTP (resp. DBTP). Conversely, a binary instance can satisfy BTP but not DBTP. This case is illustrated in

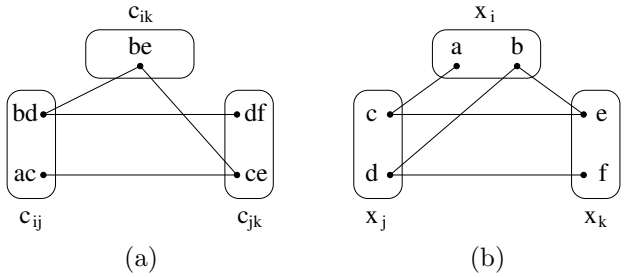


Figure 2: An instance which satisfies DBTP (a) but not BTP (b).

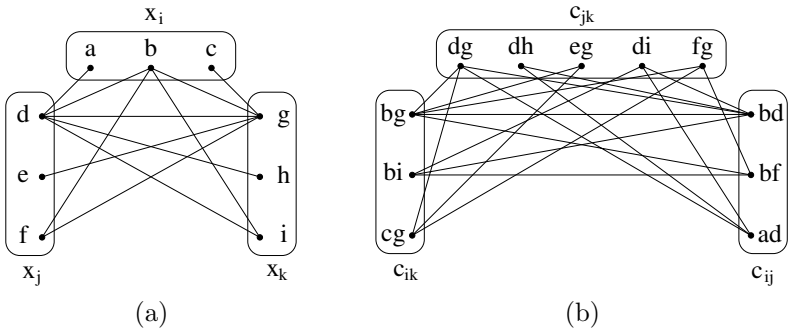


Figure 3: An instance satisfying BTP (a) but not DBTP (b).

Figure 3. Indeed, BTP holds w.r.t. the ordering $x_i < x_j < x_k$ while DBTP does not hold because $\langle (bg, dg), (bg, bf), (dg, ad) \rangle$, $\langle (bd, dg), (bd, bi), (dg, cg) \rangle$ and $\langle (bd, bg), (bd, dh), (bg, eg) \rangle$ are broken triangles (in broken lines in Figure 3) respectively on c_{ij}, c_{ik} and c_{jk}. Theorem 1 is deduced from these examples.

Theorem 1 *Let P be a binary CSP instance.*

- *P satisfies DBTP $\not\Rightarrow$ P satisfies BTP,*
- *P satisfies BTP $\not\Rightarrow$ P satisfies DBTP.*

This first theorem shows that DBTP is then not a generalization of BTP to non-binary CSPs.

We now prove that the class of CSP instances which satisfy DBTP is tractable, thanks to the two next lemmas, whose proofs exploit the approach proposed in [8].

Lemma 1 *Any CSP instance P which satisfies DBTP w.r.t. the constraint ordering $\prec$ can be solved in $O(e^2.r.\rho^2)$.*

Proof: The first step consists in building the dual of P, what can be achieved in $O(e^2.r.\rho^2)$. Then, as the dual of P is BTP, we know that it can be solved in $O(e^2.\rho^2)$ [8]. Hence, the overall complexity is $O(e^2.r.\rho^2)$. □

Lemma 2 expresses that the constraint ordering $\prec$ related to DBTP may be computed (if any) in polynomial time.

Lemma 2 *Given any CSP instance P, determining if a constraint ordering $\prec$ s.t. P is DBTP w.r.t. $\prec$ exists (and finding it if any) can be achieved in polynomial time.*

Proof: A possible algorithm consists in computing first the dual of P and then determining if an ordering $\prec$ s.t. the dual of P is BTP exists like in [8]. Both steps are polynomial (see the previous proof and [8]). Hence, the overall complexity is polynomial. □

The two previous lemmas allow to establish the tractability of DBTP.

Theorem 2 *$DBTP$ is a tractable class.*

We now present an alternative and equivalent characterization of DBTP:

Theorem 3 *A CSP instance $P = (X, D, C)$ satisfies the DBTP property w.r.t. the constraint ordering $\prec$ iff for all triples of constraints (c_i, c_j, c_k) s.t. $c_i \prec c_j \prec c_k$, for all $t_i \in R(c_i)$, $t_j \in R(c_j)$ and $t_k, t'_k \in R(c_k)$ s.t.*

- $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$
- $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$
- $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$

then

- *either* $t'_k[S(c_i) \cap S(c_k)] = t_i[S(c_i) \cap S(c_k)]$
- *or* $t_j[S(c_j) \cap S(c_k)] = t_k[S(c_j) \cap S(c_k)]$.

Proof: P satisfies DBTP w.r.t. $\prec$

$\Leftrightarrow$ P^d satisfies BTP w.r.t. $\prec$

$\Leftrightarrow$ for all triples of variables (x^d_i, x^d_j, x^d_k) s.t. $x^d_i \prec x^d_j \prec x^d_k$, for all $t_i \in d^d_i$, $t_j \in d^d_j$ and $t_k, t'_k \in d^d_k$ s.t. $(t_i, t_j) \in R(c^d_{ij})$, $(t_i, t_k) \in R(c^d_{ik})$ and $(t_j, t'_k) \in R(c^d_{jk})$ then either $(t_i, t'_k) \in R(c^d_{ik})$ or $(t_j, t_k) \in R(c^d_{jk})$

$\Leftrightarrow$ for all triples of constraints (c_i, c_j, c_k) s.t. $c_i \prec c_j \prec c_k$, for all $t_i \in R(c_i)$, $t_j \in R(c_j)$ and $t_k, t'_k \in R(c_k)$ s.t. $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$, $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$ and $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$ then either $t'_k[S(c_i) \cap S(c_k)] = t_i[S(c_i) \cap S(c_k)]$ or $t_j[S(c_j) \cap S(c_k)] = t_k[S(c_j) \cap S(c_k)]$. □

In order to illustrate this characterization, let us consider the CSP depicted in Figure 4 (a). Clearly, from the micro-structure of its dual (see Figure 4 (b)), we can deduce that this instance is DBTP w.r.t. the ordering $c_{ij} \prec c_{ik} \prec c_{jk}$ but not w.r.t. $c_{jk} \prec c_{ik} \prec c_{ij}$. Now, if we take into account the alternative characterization, we obtain the same conclusions. Indeed, regarding the ordering $c_{ij} \prec c_{ik} \prec c_{jk}$, there is a single quadruple of tuples $(t_{ij}, t_{ik}, t_{jk}, t'_{jk})$ such that $t_{ij}[\{x_i\}] = t_{ik}[\{x_i\}]$, $t_{ij}[\{x_j\}] = t_{jk}[\{x_j\}]$ and $t_{ik}[\{x_k\}] = t'_{jk}[\{x_k\}]$, namely $t_{ij} = (a, c)$, $t_{ik} = (a, e)$, $t_{jk} = (c, e)$ and $t'_{jk} = (d, e)$. As we have

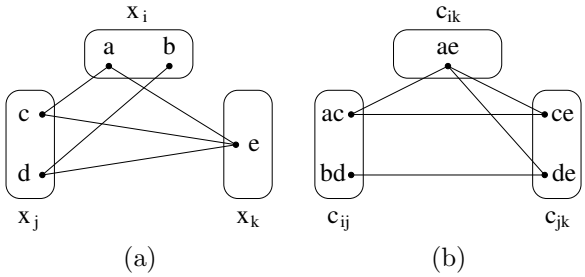


Figure 4: An instance satisfying DBTP with its micro-structure (a) and the micro-structure of its dual (b).

$t_{ik}[\{x_k\}] = t_{jk}[\{x_k\}]$, the instance is DBTP w.r.t. $c_{ij} \prec c_{ik} \prec c_{jk}$. In contrast, for the ordering $c_{jk} \prec c_{ik} \prec c_{ij}$, the quadruple of tuples $(t_{jk}, t_{ik}, t_{ij}, t'_{ij})$ with $t_{jk} = (d, e)$, $t_{ik} = (a, e)$, $t_{ij} = (b, d)$ and $t'_{ij} = (a, c)$ is such that $t_{jk}[\{x_k\}] = t_{ik}[\{x_k\}]$, $t_{jk}[\{x_j\}] = t_{ij}[\{x_j\}]$ and $t_{ik}[\{x_i\}] = t'_{ij}[\{x_i\}]$ but we have $t_{jk}[\{x_j\}] \neq t'_{ij}[\{x_j\}]$ and $t_{ik}[\{x_i\}] \neq t_{ij}[\{x_i\}]$. So the instance cannot be DBTP w.r.t. $c_{jk} \prec c_{ik} \prec c_{ij}$.

The alternative characterization introduced in Theorem 3 makes possible the recognition of DBTP instances directly by exploiting the tuples of relations without building the dual instance, what may be of significant interest from a practical viewpoint. We discuss this issue in Section 6.

2.2 Conservation by filtering and its consequences on solving

A filtering consistency ϕ is a function which associates to each CSP instance P an equivalent instance $\phi(P)$ (i.e. an instance which has the same solution set as P) by deleting values and/or tuples which cannot appear in a solution of P (see [3] for more details). Filtering consistencies are commonly exploited before or during the solving in order to simplify the instances. At present, we wonder what the DBTP property becomes when applying a filtering consistency.

Definition 5 *A class $\mathcal{C}$ of CSP instances is said **conservative** w.r.t. a filtering consistency ϕ if it is closed under ϕ, that is, if the instance obtained after the application of ϕ belongs to the class $\mathcal{C}$.*
*A property is said **conservative** if it defines a conservative class of instances.*

Property 1 *DBTP is conservative for any filtering consistency which only removes values from domains or tuples from existing relations.*

Proof: Let us consider a CSP instance P satisfying DBTP w.r.t. a given constraint ordering. The removal of a value from the domain of a variable x of P induces the deletion of some tuples for the constraints whose scope contains x. In other words, it implies the deletion of some values for the variables of the dual of P. On the other part, the deletion of some tuples is equivalent to remove some values from domains of some dual variables. Therefore, in both cases, the deletions of values or tuples in the original instance lead to remove values of the dual variables. As BTP is conservative w.r.t. domain filtering consistencies, the dual

of P after these removals still satisfies BTP. Hence, P is still DBTP. □

For instance, this property holds for any domain filtering consistency (e.g (Generalized) Arc-Consistency or Path Inverse Consistency [4]) applied on the original instances or their dual. In particular, it is the case for the pairwise-consistency [18] (introduced in the field of Relational Databases Theory [2]) which is equivalent to applying arc-consistency on the dual instance [18]. We now recall the definitions of arc and pairwise-consistency.

Definition 6 (Arc-Consistency) *Given a CSP instance $P = (X, D, C)$, a value $v_i \in d_i$ is arc-consistent w.r.t. $c \in C$ iff there exists a valid tuple $t \in R(c)$ s.t. $t[\{x_i\}] = v_i$. A domain d_i is arc-consistent w.r.t. c iff $d_i \neq \emptyset$ and $\forall v_i \in d_i$, v_i is arc-consistent w.r.t. c. P is arc-consistent iff $\forall d_i \in D$, d_i is arc-consistent w.r.t. all $c \in C$.*

For example, the instance described in Figure 5(a) is arc-consistent while one depicted in Figure 4(a) is not arc-consistent since, notably, the value b of d_i is not arc-consistent w.r.t. the constraint c_{ik}. Hence, enforcing the arc-consistency on this instance will delete the values b and d in order to make the instance arc-consistent.

Definition 7 (Pairwise-Consistency [18]) *A CSP instance $P = (X, D, C)$ is **pairwise-consistent** iff $\forall\ 1 \leq i \leq e$, $R(c_i) \neq \emptyset$ and $\forall\ 1 \leq i < j \leq e$, $R(c_i)[S(c_i) \cap S(c_j)] = R(c_j)[S(c_i) \cap S(c_j)]$.*

For example, the instance of Figure 5(a) is pairwise-consistent while one depicted in Figure 4(a) is not pairwise-consistent since, notably, $R(c_{ij})[\{x_i\}] \neq R(c_{ik})[\{x_i\}]$ ($R(c_{ij})[\{x_i\}] = \{a, b\}$ and $R(c_{ik})[\{x_i\}] = \{a\}$). Hence, enforcing the pairwise-consistency on this instance will delete the tuples (b, d) and (d, e).

We now investigate the consequence of Property 1 on the solving of instances which are DBTP. Like MAC [26] maintains the arc-consistency (denoted AC) at each step of the search, we define MPWC (for Maintaining PairWise Consistency) as the algorithm corresponding to maintain the pairwise-consistency at each step.

Theorem 4 *If a CSP instance P satisfies DBTP, then MPWC solves P in polynomial time w.r.t. any ordering.*

Proof: As the pairwise-consistency on P is equivalent to the arc-consistency on the dual of P [18], the application of MPWC on P is equivalent to the application of MAC on the dual of P. Moreover, as P is DBTP, P^d is BTP and so, according to Theorem 7.6 of [8], MPWC solves P in polynomial time. □

Finally, we can derive a similar result for MAC as soon as enforcing the arc-consistency is sufficient to entail the pairwise-consistency. We say that enforcing the arc-consistency on a CSP $P = (X, D, C)$ entails the pairwise-consistency if the instance $P' = (X, D', C)$ obtained after having achieved AC is pairwise-consistent for the remaining values in the domains of D' (i.e. $\forall\ 1 \leq i \leq e$, $\exists t \in R(c_i), \forall x_{i_k} \in S(c_i), t[\{x_{i_k}\}] \in d'_{i_k}$ and $\forall\ 1 \leq i, j \leq e, i \neq j$, $\forall t_i \in R(c_i)$ *s.t.* $\forall x_{i_k} \in S(c_i), t_i[\{x_{i_k}\}] \in d'_{i_k}, \exists t_j \in R(c_j)$ *s.t.* $\forall x_{j_k} \in S(c_j), t_j[\{x_{j_k}\}] \in d'_{j_k}, t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$). Note that we consider here a particular case where the pairwise-consistency is a logical consequence of the application of the arc-consistency. In other words, we obtain the pairwise-consistency simply by enforcing the arc-consistency and so without having to enforce explicitly the pairwise-consistency. For example, the application

of the arc-consistency on the instance depicted in Figure 4 deletes the values b and d, and indirectly the tuples (b, d) and (d, e), what makes the obtained instance pairwise-consistent.

Lemma 3 *Let P be an arc-consistent CSP instance. If the instance P' obtained from P by deleting some values and enforcing AC has no empty domain and if enforcing the arc-consistency entails the pairwise-consistency, then the dual of P' is arc-consistent.*

Proof: Let us consider P and P' obtained from P by deleting some values and enforcing AC such that it has no empty domain. Since P' is arc-consistent and enforcing the arc-consistency entails the pairwise-consistency, P' is also pairwise-consistent. Thus, as the pairwise-consistency on a CSP is equivalent to the arc-consistency on its dual [18], the dual of P' is arc-consistent. □

Theorem 5 *If a CSP instance P satisfies DBTP and at each step of the search, enforcing the arc-consistency entails the pairwise-consistency, then MAC can solve P in polynomial time w.r.t. any ordering.*

Proof: If, after having enforced arc-consistency, no domain, neither relation is empty, then the resulting instance is pairwise-consistent and has a solution. According to lemma 3, the instance obtained after deleting some values and enforcing arc-consistency still remains pairwise-consistent. Therefore, when applying MAC on the original instance, we also maintain the pairwise-consistency. Moreover, as pairwise-consistency is equivalent to arc-consistency on the dual problem [18], Theorem 7.6 of [8] implies that MAC can solve P in polynomial time w.r.t. any ordering since the dual is BTP. □

The entailment of the pairwise-consistency when enforcing the arc-consistency occurs notably when any pair of constraints share at most one variable, as stated by the following lemma.

Lemma 4 (Prop. 8.1, p. 146 in [20]) *Let $P = (X, D, C)$ be a CSP instance s.t. $\forall c_i, c_j \in C, |S(c_i) \cap S(c_j)| \leq 1$. If P is arc-consistent, then it is pairwise-consistent.*

Theorem 6 *If a CSP instance $P = (X, D, C)$ s.t. $\forall c_i, c_j \in C, |S(c_i) \cap S(c_j)| \leq 1$ is arc-consistent and satisfies DBTP, then MAC can solve P in polynomial time w.r.t. any ordering.*

Proof: According to Lemma 4, when any pair of constraints share at most one variable, achieving the arc-consistency entails the pairwise-consistency. So, from Theorem 5, MAC can solve P in polynomial time w.r.t. any ordering. □

We can note that this theorem holds in particular for binary CSPs. Moreover, we can also remark that Theorem 7.6 of [8] also holds for the algorithm RFL [23] since the proof of this theorem is only based on the enforcement of the arc-consistency at each step of the search. So, it is the same for Theorems 5 and 6. Hence, any CSP instance for which DBTP holds can be solved in polynomial time by MAC or RFL without any additional work and whatever the considered variable ordering. As most solvers of the state of the art rely on either MAC or RFL, this result may explain why these solvers are efficient in practice for solving such instances as shown in Section 6.

Finally, if we consider the instances which do not satisfy the DBTP property, we can observe that the simplified instances obtained by enforcing a given filtering consistency may be DBTP. Indeed the given filtering consistency may delete all the values or tuples which prevent the original instances from being DBTP. This issue will be studied in the next section in which we compare DBTP and BTP and in Section 6 from a practical viewpoint.

3 DBTP vs BTP

We saw with Theorem 1, that even in the case of binary CSPs, BTP and DBTP classes are different. Such a result was foreseeable since, even if the original instance and its dual represent the same problem, their structure and micro-structure are quite different. This result relies on the presence of broken triangles in the micro-structure of the instance or of its dual instance. In both cases, these broken triangles often involve values which would be deleted by some filtering consistency like arc-consistency. So, as DBTP and BTP are conservative w.r.t. domain filtering consistencies, we focus our study on binary instances which satisfy arc-consistency and thus pairwise-consistency (by lemma 3). Under these assumptions, we can prove the following lemma:

Lemma 5 *Given an arc-consistent binary CSP instance $P = (X, D, C)$, if for some triple (x_i, x_j, x_k) of variables, we have a broken triangle on x_k, then we have a broken triangle on c_{ik} and one on c_{jk} for the triple (c_{ij}, c_{ik}, c_{jk}) in the dual.*

Proof: Let x_i, x_j, $x_k \in X$ s.t. $(v_i, v_j) \in R(c_{ij})$, $(v_i, v_k) \in R(c_{ik})$, $(v_j, v'_k) \in R(c_{jk})$, $(v_i, v'_k) \notin R(c_{ik})$ and $(v_j, v_k) \notin R(c_{jk})$. As P is pairwise-consistent, there are some values $v'_i \in d_i$ and $v'_j \in d_j$ s.t. $v_i \neq v'_i$, $v_j \neq v'_j$, $(v'_i, v'_k) \in R(c_{ik})$ and $(v'_j, v_k) \in R(c_{jk})$. So, $\langle((v_i, v_j), (v_i, v_k)), ((v_i, v_j), (v_j, v'_k)), ((v_i, v_k), (v'_j, v_k))\rangle$ forms a broken triangle on c_{jk} for the triple (c_{ij}, c_{ik}, c_{jk}).
Likewise for $\langle((v_i, v_j), (v_j, v'_k)), ((v_i, v_j), (v_i, v_k)), ((v_j, v'_k), (v'_i, v'_k))\rangle$ on c_{ik}. □

The presence of a broken triangle on x_k for a triple (x_i, x_j, x_k) imposes the condition $x_k < max(x_i, x_j)$ on the variable ordering $<$ (see the proof of Theorem 3.2 of [8]). Consequently, according to lemma 5, it corresponds to impose the two conditions $c_{jk} \prec max(c_{ij}, c_{ik})$ and $c_{ik} \prec max(c_{ij}, c_{jk})$ for the triple (c_{ij}, c_{ik}, c_{jk}) on the constraint ordering $\prec$. It ensues that any arc-consistent and pairwise-consistent binary instance which satisfies BTP and has two broken triangles for two different variables of a same triple of variables cannot satisfy DBTP since we will obtain all the possible broken triangles for the corresponding triple of constraints.

Conversely, we show now that a binary instance can be arc-consistent and DBTP but not BTP. For this purpose, let us consider a binary instance with 9 variables $\{x_a, x_b, \ldots, x_i\}$. We define this instance by reproducing several times a same pattern s.t. each value appearing in an instance of the pattern does not appear in any other instance. This pattern consists in a broken triangle on a variable z for a triple (x, y, z) (i.e. which imposes the condition $z < max(x, y)$ on $<$) and each value of the variables x, y and z is linked to a given value of any variable which is not involved in this triple. We reproduce this pattern 9 times s.t. the following conditions are imposed:

- $x_a < max(x_b, x_c)$,

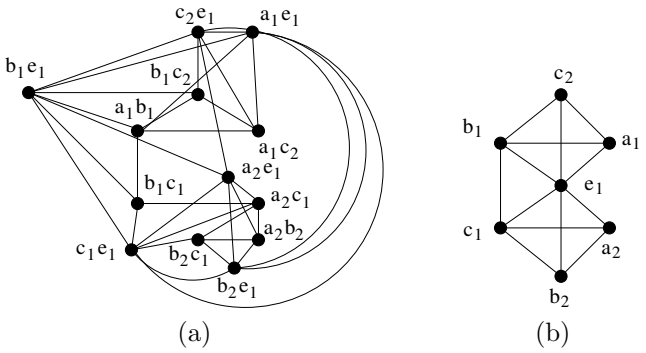


Figure 5: Part of an instance satisfying DBTP, arc-consistency and pairwise-consistency but not BTP.

- $x_b < max(x_e, x_h)$,
- $x_c < max(x_e, x_g)$,
- $x_d < max(x_a, x_g)$,
- $x_e < max(x_a, x_i)$,
- $x_f < max(x_d, x_e)$,
- $x_g < max(x_h, x_i)$,
- $x_h < max(x_b, x_d)$ and
- $x_i < max(x_c, x_f)$.

Figure 5(b) depicts this pattern for the triple (x_a, x_b, x_c), a broken triangle on x_a (corresponding to the condition $x_a < max(x_b, x_c)$) and an independent variable x_e while Figure 5 (a) describes the corresponding part of the dual instance. By doing this, the microstructure of our binary CSP or one of its dual instance have 9 connected components. We can note that this instance is not BTP because the 9 conditions make impossible the construction of a suitable variable ordering. In contrast, it is DBTP (w.r.t the ordering $c_{ab} \prec c_{ac} \prec c_{ad} \prec c_{bf} \prec c_{bh} \prec c_{ci} \prec c_{df} \prec c_{dh} \prec c_{ef} \prec c_{ei} \prec c_{gh} \prec c_{gi} \prec c_{af} \prec c_{bc} \prec c_{bd} \prec c_{ce} \prec c_{cg} \prec c_{de} \prec c_{dg} \prec c_{fi} \prec c_{hi} \prec c_{ae} \prec c_{ag} \prec c_{be} \prec c_{bg} \prec c_{cd} \prec c_{cf} \prec c_{di} \prec c_{fh} \prec c_{ai} \prec c_{bi} \prec c_{eg} \prec c_{eh} \prec c_{fg} \prec c_{ah} \prec c_{ch}$), arc-consistent and pairwise-consistent.

Now, we focus our study on acyclic CSP instances. [8] has already proved that such binary CSP instances satisfy BTP. We are going to show that this is also true for DBTP. Let $TREE$ be the set of binary CSP instances whose constraint graph is acyclic.

Theorem 7 $TREE \subsetneq DBTP$.

Proof: Let $DUAL\text{-}TREE$ be the set of binary instances which are the dual of instances from $TREE$. As shown in [8], $DUAL\text{-}TREE \subsetneq BTP$. Hence $TREE \subsetneq DBTP$. □

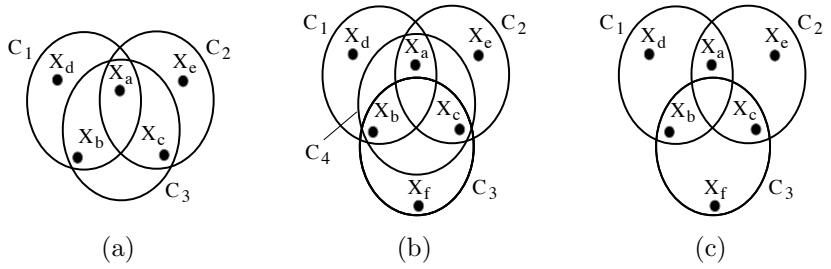


Figure 6: A β-acyclic hypergraph (a), an α-acyclic and β-cyclic hypergraph (b) and an α and β-cyclic hypergraph (c).

This result can be extended to CSP instances of arbitrary arity. For this, we must consider the notion of cyclicity in hypergraphs, for which different degrees of cyclicity have been defined [2, 13]. Here, we are interested by α-acyclicity and β-acyclicity. We will first prove that β-acyclic CSPs satisfy DBTP and later, we will show it is not the case for α-acyclic CSPs. We first recall the definition of β-acyclicity of (constraint) hypergraph.

Definition 8 ([17]) *$H = (X, C)$ is a β-acyclic hypergraph iff it has no Graham cycle. A sequence $(c_1, ..., c_m, c_{m+1})$ with $m \geq 3$ s.t. $(c_1, ..., c_m)$ are distinct and $c_1 = c_{m+1}$ is a Graham cycle if each $\Delta_i = S(c_i) \cap S(c_{i+1})$ $(1 \leq i \leq m)$ is nonempty, and whenever $i \neq j$, Δ_i and Δ_j are incomparable (i.e. $\Delta_i \not\subseteq \Delta_j$ and $\Delta_j \not\subseteq \Delta_i$).*

Figure 6(a) depicts a β-acyclic hypergraph while we have two β-cyclic hypergraphs in (b) and (c).

It has been recently shown in [9] that β-acyclic hypergraphs can be defined by applying the two following rules that yield the empty hypergraph:

(1) If a hyperedge is empty, we remove it from C.

(2) If a vertex is a nest point (i.e. the set of hyperedges containing it is a chain for the inclusion relation), then we remove it from H (i.e. from X and from the hyperedges that contain it).

Theorem 8 ([9]) *A hypergraph H is β-acyclic if and only if, after applying the two rules successively until none can be applied, we obtain the empty hypergraph.*

Using these definitions, we can now establish the next theorem:

Theorem 9 *Given a CSP instance (X, D, C), there exists a constraint ordering $\prec$, s.t. $\forall c_i, c_j, c_k \in C$ s.t. $c_i \prec c_j \prec c_k$, we have $S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k)$ or $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$ if and only if (X, D, C) has a β-acyclic constraint hypergraph.*

Proof: ($\Rightarrow$) By contraposition. So we show that if the constraint hypergraph (X, C) is β-cyclic, then, there is no constraint ordering.

Consider a constraint hypergraph (X, C) which is β-cyclic. So, it has a Graham cycle, denoted by the sequence of hyperedges $(c_1, ..., c_m, c_{m+1})$. Consider an arbitrary constraint

ordering $\prec$. Necessarily, among the constraints of this cycle, there is a maximum constraint c_k w.r.t. the ordering $\prec$. Consider its two neighbors in the cycle, denoted c_i and c_j (with $c_i, c_j \prec c_k$). By definition of Graham cycles, we know that $S(c_i) \cap S(c_k)$ and $S(c_j) \cap S(c_k)$ are incomparable. So, we have neither $S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k)$ nor $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$, and then no suitable constraint ordering $\prec$ exists.

($\Leftarrow$) Here, we use Theorem 8. So, given a CSP instance (X, D, C) which admits a constraint ordering $\prec$ and has a β-acyclic constraint hypergraph H, we will show that:

(1) A hyperedge $S(c_i)$ is empty iff H without $S(c_i)$ admits an ordering and is β-acyclic.

(2) A vertex x of H is a nest point such H admits an ordering iff H without x admits an ordering and is β-acyclic.

It is immediate to see that the property holds by applying the rule (1). So, consider the rule (2). Assume that for a hypergraph H we have an ordering $\prec$. So, $\forall c_i, c_j, c_k \in C$ s.t. $c_i \prec c_j \prec c_k$, we have $S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k)$ or $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$. We have five cases to consider:

1. $x \notin S(c_i) \cup S(c_j) \cup S(c_k)$: thus after the deletion of x, neither $S(c_i) \cap S(c_k)$ nor $S(c_j) \cap S(c_k)$ have changed. So, the property holds.

2. $x \in S(c_i) \cap S(c_j) \cap S(c_k)$: thus after the deletion of x, it disappears from each intersection $S(c_i) \cap S(c_k)$ and $S(c_j) \cap S(c_k)$, and thus, the property holds.

3. x belongs to only one set $S(c_i)$ or $S(c_j)$ or $S(c_k)$: so x belongs to no intersection and thus, the property holds after the deletion.

4. $x \in S(c_i) \cap S(c_j)$ and $x \notin S(c_k)$: so x belongs neither to $S(c_i) \cap S(c_k)$, nor to $S(c_i) \cap S(c_k)$ and thus, the property holds after the deletion.

5. $x \in S(c_i) \cap S(c_k)$ and $x \notin S(c_j)$ (or symmetrically $x \in S(c_j) \cap S(c_k)$ and $x \notin S(c_i)$): so before the deletion, we have necessarily $S(c_i) \cap S(c_k) \not\subseteq S(c_j) \cap S(c_k)$ and $S(c_j) \cap S(c_k) \subsetneq S(c_i) \cap S(c_k)$. Thus, after the deletion of x, we have at least $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$

So we have shown that if we can delete the whole hypergraph, which does not contradict the property on the ordering, necessarily, the first hypergraph is β-acyclic and admits a suitable constraint ordering. □

We can note that this theorem explains why the condition enunciated in lemma 4.6 of [8] (recalled below) holds independently from the scope of the constraints.

Lemma 4.6. of [8] *Let P be a CSP instance (of arbitrary arity) with constraint scopes $S(c_1), S(c_2), \ldots, S(c_e)$, where the constraints allow all combinations of values from some fixed domain D. The dual instance of P, with corresponding variables $1, 2, \ldots, e$, has the BTP property with respect to some ordering $<$ if, and only if, for all triples $S(c_i), S(c_j), S(c_k)$ with $i < j < k$ we have*

$$S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k) \text{ or } S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k).$$

Moreover, if this condition holds, then the dual of any instance P' with the same constraint scopes also has the BTP property with respect to $<$.

This lemma and the previous theorem allow us to obtain Theorem 10 where β-$ACYCLIC$ is the set of CSP instances whose constraint (hyper)graph is β-acyclic.

Theorem 10 *β-$ACYCLIC \subsetneq DBTP$.*

Proof: According to Theorem 9, we know that for any β-acyclic CSP instance (X, D, C), there exists a constraint ordering $\prec$, s.t. $\forall c_i, c_j, c_k \in C$ s.t. $c_i \prec c_j \prec c_k$, we have $S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k)$ or $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$. Moreover, according to lemma 4.6 of [8], any CSP instance satisfying the latter condition has a BTP dual. Hence β-$ACYCLIC \subsetneq DBTP$. □

Note that the equivalence in this lemma 4.6 only holds for the reverse direction. However this does not endanger the proof of Theorem 10. Clearly, it suffices to consider a binary instance with 3 monovalent variables (i.e. variables whose domain contains a single value) pairwise connected (each constraint allows the single tuple). Its dual satisfies BTP while the constraint graph is not β-acyclic.

Now, we show that if α-$ACYCLIC$ is the set of CSP instances whose constraint (hyper)graph is α-acyclic, then the sets α-$ACYCLIC$ and $DBTP$ are incomparable. So, we recall that α-acyclicity of (constraint) hypergraphs can be defined using the "running intersection property" [2], namely:

Definition 9 *(X, C) is an α-acyclic hypergraph iff there exists an ordering $(c_1, ..., c_e)$ s.t. $\forall k, 1 < k \leq e, \exists j < k, (S(c_k) \cap \bigcup_{i=1}^{k-1} S(c_i)) \subseteq S(c_j)$.*

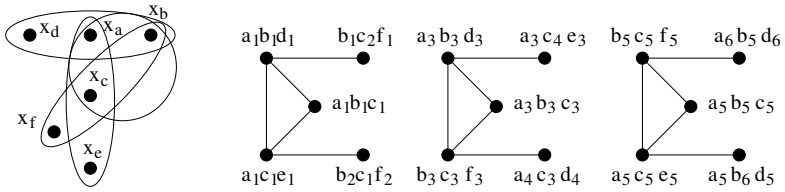
Let us consider a CSP instance with six variables $x_a, \ldots x_f$ and four constraints whose scope are respectively $\{x_a, x_b, x_c\}$, $\{x_a, x_b, x_d\}$, $\{x_a, x_c, x_e\}$ and $\{x_b, x_c, x_f\}$. Figure 7 depicts its constraint hypergraph (a) and the micro-structure of its dual (b). We can note that this instance is α-acyclic but does not satisfy DBTP since no suitable constraint ordering exist. Moreover, it is well known that β-$ACYCLIC \subsetneq \alpha$-$ACYCLIC$ [13]. In this paper, several equivalent definitions of β-acyclicity are given, the most simple one indicating that a hypergraph is β-acyclic if and only if every one of its subhypergraphs is α-acyclic (a *subhypergraph* of a hypergraph is defined as a subset, not necessarily proper, of its set of hyperedges). For example, in Figure 6 (b), we have a β-cyclic hypergraph which is α-acyclic while in (c), we have an α-cyclic hypergraph which is necessary β-cyclic.
Hence, if we denote $A \perp B$ two tractable classes which are incomparable (i.e. neither $A \subseteq B$, nor $B \subseteq A$), we obtain the following theorem:

Theorem 11 *α-$ACYCLIC \cap DBTP \neq \emptyset$ and α-$ACYCLIC \perp DBTP$.*

Through this section, we have established:

Theorem 12 *$BTP \cap DBTP \neq \emptyset$ and $BTP \perp DBTP$.*

In the next section, we study the link between DBTP and some other well known tractable classes.



(a) (b)

Figure 7: An α-acyclic CSP instance (a) but which does not satisfy DBTP (b).

4 DBTP vs some tractable classes

4.1 For binary CSPs

As DBTP and BTP are two different classes, we first focus on some tractable classes included in BTP. These classes whose definitions are recalled below rely on restricted constraint languages.

Definition 10 (Row-convex [1]) *A binary CSP instance $P = (X, D, C)$ is said **row-convex** w.r.t. a variable ordering $<$ and a value ordering, if, for each constraint c_{ij} of C with $x_i < x_j$, $\forall v_i \in d_i$, $\{v_j \in d_j | (v_i, v_j) \in R(c_{ij})\} = [a_j..b_j]$ for some a_j, $b_j \in d_j$ where $[a_j..b_j]$ denotes the values belonging to d_j between a_j and b_j w.r.t. the value ordering. We denote RC the set of row-convex instances.*

Definition 11 (0-1-all [7]) *A binary CSP instance $P = (X, D, C)$ is said **0-1-all** if for each constraint c_{ij} of C, for each value $v_i \in d_i$, c_{ij} satisfies one of the following conditions:*

- *(ZERO) for any value $v_j \in d_j, (v_i, v_j) \notin R(c_{ij})$,*
- *(ONE) there is a unique value $v_j \in d_j$ such as $(v_i, v_j) \in R(c_{ij})$,*
- *(ALL) for any value $v_j \in d_j, (v_i, v_j) \in R(c_{ij})$.*

We denote ZOA the set of instances which are 0-1-all.

Definition 12 (Renamable right monotone [8]) *A binary CSP instance $P = (X, D, C)$ is said **renamable right monotone** w.r.t. a variable ordering $<$ if, for $2 \leq j \leq n$, each domain d_j can be ordered by $\preccurlyeq_j$ s.t. for each constraint c_{ij} of C with $x_i < x_j$, $\forall v_i \in d_i, v_j, v'_j \in d_j$, if $(v_i, v_j) \in R(c_{ij})$ and $v_j \preccurlyeq_j v'_j$ then $(v_i, v'_j) \in R(c_{ij})$.*
We denote RRM the set of these instances.

The next theorem shows that these tractable classes share some instances with DBTP but are different.

Theorem 13 *$RC \cap DBTP \neq \emptyset$ and $RC \perp DBTP$.*
$ZOA \cap DBTP \neq \emptyset$ and $ZOA \perp DBTP$.
$RRM \cap DBTP \neq \emptyset$ and $RRM \perp DBTP$.

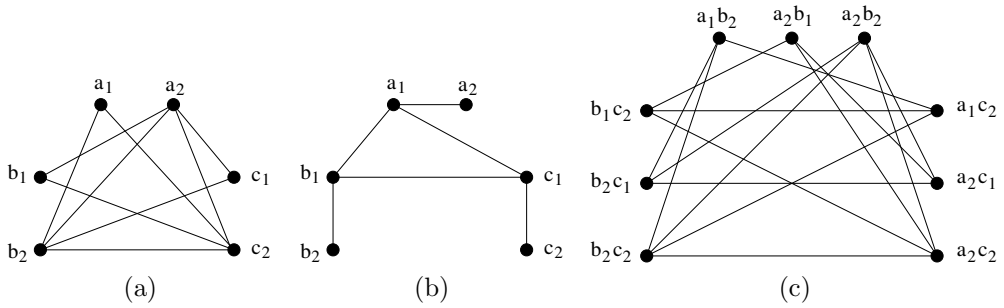


Figure 8: A CSP instance which is RC, ZOA and RRM and has a chordal micro-structure (a) and a chordal complement of micro-structure (b), but which is not DBTP (c).

Proof: If we consider the binary CSP instance of Figure 8(a), it is 0-1-all, row-convex, and renamable right monotone w.r.t. the lexicographic value and variable orderings. However, as shown in Figure 8(c), this instance is not DBTP. Conversely, any non-binary DBTP instance cannot belong to RC, ZOA or RRM.

In order to prove that $DBTP$ intersects RC, ZOA and RRM, it is sufficient to consider a monovalent and consistent binary CSP instance with three variables and three constraints since such an instance satisfies both $DBTP$, RC, ZOA and RRM. □

Some tractable classes are related to some graphical features of their micro-structure. This is the case of the class of instances which have a chordal micro-structure [15, 21]:

Definition 13 (Chordal micro-structure) *A graph is said* ***chordal*** *[16] if it has no cycle of length greater than 3 without a chord (i.e. an edge joining two non-consecutive vertices in the cycle).*
We denote CM the set of instances which have a ***chordal micro-structure****.*

Theorem 14 *$CM \cap DBTP \neq \emptyset$ and $CM \perp DBTP$.*

Proof: Any monovalent and consistent binary CSP instance has a chordal micro-structure and is DBTP. So the intersection is not empty.

Consider the instance depicted in Figure 8(a). It has a chordal micro-structure but is not DBTP. Conversely, any non-binary DBTP instance cannot belong to CM. □

Likewise, the instances for which the complement of their micro-structure is chordal also form a tractable class [6]:

Definition 14 (Chordal complement of micro-structure) *The complement of a graph $G = (X, E)$ is the graph (X, E') with $E' = \{\{x, y\} | x, y \in X \text{ s.t. } \{x, y\} \notin E\}$.*
We denote CCM the set of instances for which the complement of their micro-structure is chordal.

Theorem 15 *$CCM \cap DBTP \neq \emptyset$ and $CCM \perp DBTP$.*

Proof: Any monovalent and consistent binary CSP instance has a chordal complement of micro-structure and is DBTP. So the intersection is not empty.

Consider the instance depicted in Figure 8(a). The complement of its micro-structure, depicted in Figure 8(b), is chordal but this instance is not DBTP. Conversely, any non-binary DBTP instance cannot belong to CCM. □

As chordal graphs are also perfect graphs, we can derive a similar result for the class of instances whose micro-structure is a perfect graph [27]:

Definition 15 (Perfect micro-structure) *A graph is said* ***perfect*** *[16] if it contains no cycle, neither the complement of a cycle with an odd length greater than 4.*
We denote PM the set of instances which have a ***perfect micro-structure****.*

Theorem 16 *$PM \cap DBTP \neq \emptyset$ and $PM \perp DBTP$.*

The next class relies on the number of maximal cliques of the micro-structure:

Definition 16 (Maximal clique bounded [10]) *A CSP instance P is said* ***maximal clique bounded*** *if the number of maximal cliques in its micro-structure is polynomial w.r.t the size of P.*
We denote CL the set of these instances.

Theorem 17 *$CL \cap DBTP \neq \emptyset$ and $CL \perp DBTP$.*

Proof: Any monovalent and consistent binary CSP has a single maximal clique and is DBTP. So the intersection is not empty.

Consider any binary CSP instance s.t. its micro-structure has a polynomial number of maximal cliques. We add to such a CSP instance additional variables with additional values and additional constraints corresponding to the instance depicted in Figure 1, s.t. these values are not compatible with those of the first part of this instance. So, it has a polynomial number of maximal cliques in its micro-structure but is not DBTP. Conversely, any non-binary DBTP instance cannot belong to CL. □

Regarding classes based on restricted structures, we have proved in Theorem 7 that $TREE \subsetneq DBTP$.

4.2 For CSPs of arbitrary arity

We first consider some known tractable classes based on restricted constraint languages like the max-closed class.

Definition 17 (Max-closed [19]) *A CSP instance $P = (X, D, C)$ is said* ***max-closed*** *if for each constraint c of arity r_c, $\forall (v_1, v_2, \ldots, v_{r_c}), (v'_1, v'_2, \ldots, v'_{r_c}) \in R(c)$, $(max(v_1, v'_1), max(v_2, v'_2), \ldots, max(v_{r_c}, v'_{r_c})) \in R(c)$.*
We denote MC the set of max-closed instances.

Theorem 18 *$MC \cap DBTP \neq \emptyset$ and $MC \perp DBTP$.*

Proof: The proof of $MC \cap DBTP \neq \emptyset$ and $MC \not\subseteq DBTP$ is similar to one of Theorem 13. Regarding $DBTP \not\subseteq MC$, any CSP instance having two variables and one binary constraint is DBTP but not necessarily max-closed. □

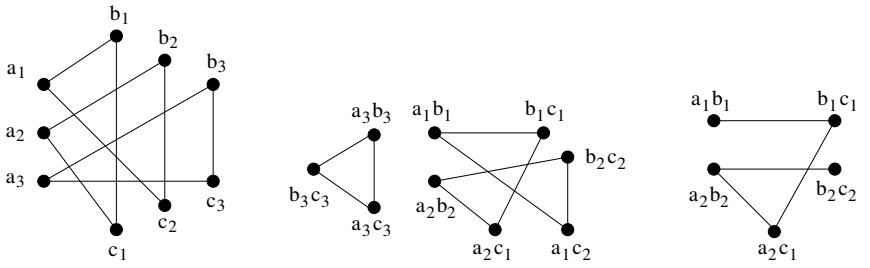


Figure 9: An incrementally functional instance (a) but which does not satisfy DBTP (b). An instance which is DBTP but not triangular (c).

Definition 18 (Incrementally functional [5]) *A CSP instance $P = (X, D, C)$ is said **incrementally functional** if there exists a variable ordering $<$ s.t. for $1 \leq i < n$, each solution of $P[\{x_1, \ldots, x_i\}]$ extends to at most one solution of $P[\{x_1, \ldots, x_{i+1}\}]$ where, for $X' \subseteq X, P[X']$ denotes the CSP instance (X', D', C') where $D' = \{d_i | x_i \in X'\}$ and $C' = \{c' | c \in C$ s.t. $S(c) \cap X' \neq \emptyset, S(c') = S(c) \cap X'$ and $R(c') = \{t[S(c')] | t \in R(c)\}\}$. We denote $IFUN$ the set of these instances.*

Theorem 19 *$IFUN \cap DBTP \neq \emptyset$ and $IFUN \perp DBTP$.*

Proof: In order to prove that the intersection is not empty, we consider a CSP instance with four monovalent variables $x_1, \ldots, x_4$ and three ternary constraints c_1, c_2 and c_3 s.t. $S(c_1) = \{x_1, x_2, x_3\}$, $R(c_1) = \{(v_1, v_2, v_3)\}$, $S(c_2) = \{x_1, x_2, x_4\}$, $R(c_2) = \{(v_1, v_2, v_4)\}$, $S(c_3) = \{x_2, x_3, x_4\}$ and $R(c_3) = \{(v_2, v_3, v_4)\}$. This instance is incrementally functional (using the numeration of variables as ordering) and DBTP.

The instance of Figure 9(a) is incrementally functional but not DBTP (b). Conversely, any DBTP instance having several solutions cannot be incrementally functional. □

Definition 19 (Dual maximal clique bounded [10]) *A CSP instance P is said **dual maximal clique bounded (DMCB)** if the number of maximal cliques in the micro-structure of its dual instance is polynomial w.r.t. the size of P.*
We denote DCL the set of these instances.

Theorem 20 *$DCL \cap DBTP \neq \emptyset$ and $DCL \perp DBTP$.*

Proof: Let us consider the first instance defined in the proof of Theorem 19. The micro-structure of its dual instance has a single maximal clique and the instance is DBTP. So, the intersection is not empty. Regarding the instance depicted in Figure 9(b), it has a polynomial number of maximal cliques in the micro-structure of its dual instance but is not DBTP. Conversely, a binary instance whose constraint graph is a star and for which each domain has several values is DBTP but has an unbounded number of maximal cliques in the micro-structure of its dual. □

Now we introduce a new tractable class based on a constraint language restriction.

Definition 20 (Triangular) *A CSP instance $P = (X, D, C)$ is said **triangular** w.r.t. a constraint ordering $\prec$ iff $\forall c_i, c_j, c_k$, $c_i \prec c_j \prec c_k$, $\forall t_i \in R(c_i), t_j \in R(c_j), t_k \in R(c_k)$, if $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$ and $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$ then, $t_j[S(c_j) \cap S(c_k)] = t_k[S(c_j) \cap S(c_k)]$.*
We denote TR the set of triangular instances.

Theorem 21 *If a CSP instance is triangular w.r.t. $\prec$, then it satisfies DBTP w.r.t. $\prec$.*

Proof: Assume that P is triangular but not DBTP. So, there exist three constraints c_i, c_j and c_k, $c_i \prec c_j \prec c_k$, $t_i \in R(c_i)$, $t_j \in R(c_j)$ and $t_k, t'_k \in R(c_k)$ s.t. $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$, $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$, $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$, $t'_k[S(c_i) \cap S(c_k)] \neq t_i[S(c_i) \cap S(c_k)]$ and $t_j[S(c_j) \cap S(c_k)] \neq t_k[S(c_j) \cap S(c_k)]$.
As P is triangular w.r.t. $\prec$, we must have $t'_k[S(c_i) \cap S(c_k)] = t_i[S(c_i) \cap S(c_k)]$ and $t_j[S(c_j) \cap S(c_k)] = t_k[S(c_j) \cap S(c_k)]$, what is not possible since P does not satisfy DBTP. □

Theorem 22 $TR \subsetneq DBTP$.

Proof: Theorem 21 shows that $TR \subseteq DBTP$. The instance depicted in Figure 9(c) satisfies DBTP but is not triangular. □

Regarding classes based on restricted structures, we have proved in Theorems 10 and 11 that β-$ACYCLIC \subsetneq DBTP$, α-$ACYCLIC \cap DBTP \neq \emptyset$ and α-$ACYCLIC \perp DBTP$. Another important tractable class based on restricted structure is related to the tree-width. We first recall the notion of tree-decomposition of graphs [24].

Definition 21 (Tree-decomposition) *A **tree-decomposition** of a graph $G = (X, E)$ is a pair (N, T) where $T = (I, F)$ is a tree with nodes I and edges F and $N = \{N_i : i \in I\}$ is a family of subsets of X, s.t. each subset N_i is a node of T and verifies:*

(i) $\cup_{i \in I} N_i = X$,

(ii) for each edge $\{x, y\} \in E$, there exists $i \in I$ with $\{x, y\} \subseteq N_i$, and

(iii) for all $i, j, k \in I$, if k is in a path from i to j in T, then $N_i \cap N_j \subseteq N_k$.

*The width w of a tree-decomposition (N, T) is equal to $max_{i \in I}|N_i| - 1$. The **tree-width** w of G is the minimal width over all the tree-decompositions of G.*

Classically, this definition is extended to hypergraphs by considering the notion of primal graphs. The primal graph of a hypergraph (X, E) is the graph (X, E') where $E' = \{\{x, y\} | \exists e \in E \text{ s.t. } x, y \in e\}$.

Definition 22 (Bounded tree-width) *Let k be a fixed positive integer. The class BTW_k is the set of the instances whose tree-width is bounded by k.*

Theorem 23 $BTW_1 \subsetneq DBTP$.
For $k > 1$, $BTW_k \cap DBTP \neq \emptyset$ and $BTW_k \perp DBTP$.

Proof: It is well known that BTW_1 is the set of tree-structured binary CSP instances. So according to Theorem 7, we have $BTW_1 \subsetneq DBTP$.

For $k > 1$, as $BTW_1 \subsetneq BTW_k$, the intersection $BTW_k \cap DBTP$ is not empty. Now, let us consider an instance having n variables with $n \geq 3$, whose tree-width is bounded by a constant $k \geq 2$ and which contains the subproblem depicted in Figure 9(a). This instance has a bounded tree-width but does not satisfy DBTP. Conversely, any instance having n variables and one constraint of arity n is DBTP but has an unbounded tree-width. Hence $BTW_k \perp DBTP$. □

5 Links between (D)BTP and Directional (Hyper-)k-Consistency

In this section, we study the links existing between (D)BTP and Directional (Hyper-)k-Consistency. Such a study is quite natural, since, as pointed in [8], BTP is a weaker form of hyper-3-consistency [22]. We recall the definition of the Hyper-k-Consistency:

Definition 23 (Hyper-k-Consistency [22]) *Given a CSP instance $P = (X, D, C)$ and an integer k s.t. $1 \leq k \leq e$, P satisfies the hyper-k-consistency if, for every subset $\{c_1, c_2, \ldots c_{k-1}, c_k\}$ of k constraints, we have*

$$\bowtie_{i=1}^{k-1} R(c_i)[(\bigcup_{i=1}^{k-1} S(c_i)) \cap S(c_k)] \subseteq R(c_k)[(\bigcup_{i=1}^{k-1} S(c_i)) \cap S(c_k)]$$

As the hyper-3-consistency implies BTP, we can consider that the hyper-k-consistency is a too strong property. Hence, we define a weaker form by taking into account an ordering over the constraints:

Definition 24 (Directional Hyper-k-Consistency) *Given a CSP instance $P = (X, D, C)$, a constraint ordering $\prec$ and an integer k s.t. $1 \leq k \leq e$, P satisfies the directional hyper-k-consistency if for every subset of k constraints s.t. $c_1 \prec c_2 \prec \ldots \prec c_{k-1} \prec c_k$,*

$$\bowtie_{i=1}^{k-1} R(c_i)[(\bigcup_{i=1}^{k-1} S(c_i)) \cap S(c_k)] \subseteq R(c_k)[(\bigcup_{i=1}^{k-1} S(c_i)) \cap S(c_k)]$$

It is easy to see that directional hyper-k-consistency is a weaker form of hyper-k-consistency. For example, let us consider an instance with three constraints, c_1, c_2 and c_3 such that:

- $S(c_1) = \{x_1, x_2, x_3\}$ and $R(c_1) = \{(a, b, c)\}$,
- $S(c_2) = \{x_2, x_3, x_4\}$ and $R(c_2) = \{(b, c, d)\}$,
- $S(c_3) = \{x_1, x_4, x_5\}$ and $R(c_3) = \{(a, d, e), (a, f, e)\}$.

With the ordering $c_1 \prec c_2 \prec c_3$, this instance satisfies directional hyper-3-consistency since $R(c_1) \bowtie R(c_2)[(S(c_1) \cup S(c_2)) \cap S(c_3)] = R(c_1) \bowtie R(c_2)[\{x_1, x_4\}] = \{(a, d)\} \subseteq R(c_3)[\{x_1, x_4\}] = \{(a, d), (a, f)\}$, while hyper-3-consistency is not verified since $R(c_1) \bowtie R(c_3)[(S(c_1) \cup S(c_3)) \cap S(c_2)] = R(c_1) \bowtie R(c_3)[\{x_2, x_3, x_4\}] = \{(b, c, d), (b, c, f)\}$ which is not a subset of $R(c_2)[(S(c_1) \cup S(c_3)) \cap S(c_2)] = \{(b, c, d)\}$.

Theorem 24 *If a CSP instance P satisfies DBTP w.r.t. a constraint ordering $\prec$ and is directional hyper-k-consistent w.r.t. $\prec$ for $2 \leq k < e$, then P is directional hyper-$(k+1)$-consistent w.r.t. $\prec$.*

Proof: Assume that P is not hyper-$(k+1)$-consistent. So, there is a subset of $k+1$ constraints s.t. $c_1 \prec \ldots \prec c_k \prec c_{k+1}$, $\exists (t_1, \ldots, t_k) \in R(c_1) \times \ldots \times R(c_k), \forall t_{k+1} \in R(c_{k+1})$, $\bowtie_{i=1}^{k} t_i[(\bigcup_{i=1}^{k} S(c_i)) \cap S(c_{k+1})] \neq t_{k+1}[(\bigcup_{i=1}^{k} S(c_i)) \cap S(c_{k+1})]$.
Let us consider the k subsets of k constraints $\{c_1, \ldots c_{j-1}, c_{j+1}, \ldots c_k, c_{k+1}\}$, for $1 \leq j \leq k$. As P is directional hyper-k-consistent, for $1 \leq j \leq k$, there exists a tuple t_{k+1}^{j} of $R(c_{k+1})$ s.t. $\bowtie_{i=1, i\neq j}^{k} t_i[(\bigcup_{i=1, i\neq j}^{k} S(c_i)) \cap S(c_{k+1})] = t_{k+1}^{j}[(\bigcup_{i=1, i\neq j}^{k} S(c_i)) \cap S(c_{k+1})]$.
Consider $1 \leq j < j' \leq k$. We have two cases:

(1) $t_{k+1}^{j} = t_{k+1}^{j'}$. Then $t_{j'}[S(c_{j'}) \cap S(c_{k+1})] = t_{k+1}^{j}[S(c_{j'}) \cap S(c_{k+1})]$ and $t_j[S(c_j) \cap S(c_{k+1})] = t_{k+1}^{j'}[S(c_j) \cap S(c_{k+1})]$.
So $\bowtie_{i=1}^{k} t_i[(\bigcup_{i=1}^{k} S(c_i)) \cap S(c_{k+1})] = t_{k+1}^{j}[(\bigcup_{i=1}^{k} S(c_i)) \cap S(c_{k+1})]$ and we have a contradiction.

(2) $t_{k+1}^{j} \neq t_{k+1}^{j'}$. We have $t_j[S(c_j) \cap S(c_{j'})] = t_{j'}[S(c_j) \cap S(c_{j'})]$, $t_{j'}[S(c_{j'}) \cap S(c_{k+1})] = t_{k+1}^{j}[S(c_{j'}) \cap S(c_{k+1})]$, $t_j[S(c_j) \cap S(c_{k+1})] = t_{k+1}^{j'}[S(c_j) \cap S(c_{k+1})]$, $t_{j'}[S(c_{j'}) \cap S(c_{k+1})] \neq t_{k+1}^{j'}[S(c_{j'}) \cap S(c_{k+1})]$ and $t_j[S(c_j) \cap S(c_{k+1})] \neq t_{k+1}^{j}\ [S(c_j) \cap S(c_{k+1})]$.
Hence P does not satisfy DBTP w.r.t. $\prec$ and we have again a contradiction.

So, P is directional hyper-$(k+1)$-consistent w.r.t. the order $\prec$. □

As the pairwise-consistency corresponds to the hyper-2-consistency, we can deduce this corollary:

Corollary 1 *If a CSP instance P satisfies DBTP w.r.t. a constraint ordering $\prec$ and is directional pairwise-consistent w.r.t. $\prec$, then P is directional hyper-$(k+1)$-consistent w.r.t. $\prec$ for $1 \leq k < e$.*

So a CSP instance which is both DBTP and directional pairwise-consistent w.r.t. a given constraint ordering is consistent. It also ensues that such a CSP instance satisfies the directional hyper-3-consistency.

Moreover, as the hyper-k-consistency corresponds to the k-consistency on the dual problem, we can also derive the following theorem and corollary by achieving a similar reasoning.

Theorem 25 *If a binary CSP instance P satisfies BTP w.r.t. a variable ordering $<$ and is directional k-consistent w.r.t. $<$ for $2 \leq k < n$, then P is directional $(k+1)$-consistent w.r.t. $<$.*

Corollary 2 *If a binary CSP instance P satisfies BTP w.r.t. a variable ordering $<$ and is directional arc-consistent (DAC) w.r.t. $<$, then P is directional $(k+1)$-consistent w.r.t. $<$ for $1 \leq k < n$.*

As a consequence, a binary CSP instance which is both BTP and DAC w.r.t. a given variable ordering is consistent.

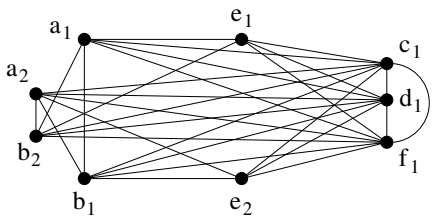


Figure 10: Part of an instance satisfying directional hyper-3-consistency but not BTP.

Regarding the positioning of the directional hyper-3-consistency w.r.t. to BTP, we can prove that the directional hyper-3-consistency does not necessarily imply BTP. For example, we can consider a binary instance with 6 variables $\{x_a, x_b, \ldots, x_f\}$. We define this instance by reproducing several times a same pattern s.t. each value appearing in an instance of the pattern does not appear in any other instance. This pattern consists in a broken triangle on a variable z for a triple (x, y, z) (i.e. which imposes the condition $z < max(x, y)$ on $<$) and each value of the variables x, y and z is linked to a given value of any variable which is not involved in this triple. We reproduce this pattern 6 times s.t. the following conditions are imposed:

- $x_a < max(x_b, x_c)$,
- $x_b < max(x_d, x_e)$,
- $x_c < max(x_e, x_f)$,
- $x_d < max(x_a, x_b)$,
- $x_e < max(x_a, x_b)$ and
- $x_f < max(x_a, x_b)$.

Figure 10 depicts this pattern for the triples (x_a, x_b, x_e), a broken triangle on x_e (corresponding to the condition $x_e < max(x_a, x_b)$) and the independent variables x_c, x_d and x_f. By doing this, the micro-structure of our binary CSP instance has 6 connected components. We can note that this instance is not BTP because the 6 conditions make impossible the construction of a suitable variable ordering. Nevertheless, it is directional hyper-3-consistent and arc-consistent.

6 DBTP from a practical viewpoint

In this section, we study the DBTP property from a practical viewpoint. First, we show that some benchmarks which are classically used for solver comparisons are DBTP. Then, for these benchmarks, we highlight the consequences on the solving efficiency.

Algorithm 1: Is_DBTP

Input: a CSP instance $P = (X, D, C)$
Output: Boolean

1 $X^o \leftarrow \{x_1^o, \ldots, x_e^o\}$
2 $D^o \leftarrow \{d_1^o, \ldots, d_e^o \mid \forall 1 \leq i \leq e, d_i^o = \{1, \ldots, e\}\}$
3 $C^o \leftarrow \emptyset$
4 **foreach** $c_i,\ c_j,\ c_k \in C$ *s.t.* $c_i \neq c_j$, $c_i \neq c_k$ *and* $c_j \neq c_k$ **do**
5 | **if** $\exists t_i \in R(c_i), t_j \in R(c_j), t_k, t'_k \in R(c_k)$ *s.t.* $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$,
6 | $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$, $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$,
7 | $t'_k[S(c_i) \cap S(c_k)] \neq t_i[S(c_i) \cap S(c_k)]$ *and* $t_j[S(c_j) \cap S(c_k)] \neq t_k[S(c_j) \cap S(c_k)]$ **then**
8 | | $C^o \leftarrow C^o \cup \{c | S(c) = \{o_i, o_j, o_k\}$ *and* $R(c) = \{o_k < max(o_i, o_j)\}\}$
9 **if** *the CSP instance* $P^o = (X^o, D^o, C^o)$ *has a solution* **then**
10 | **return** *true*
11 **else**
12 | **return** *false*

6.1 Benchmarks which are DBTP

Checking whether an instance P has the DBTP property can be achieved by testing the existence of a constraint ordering $\prec$ s.t. P is DBTP w.r.t. $\prec$. To do this, the proof of Lemma 2 suggests a method which consists in computing first the dual of P and then checking if a constraint ordering s.t. the dual of P is BTP exists like in [8]. However, in practice, computing the dual may require a prohibitive amount of memory. It is especially the case if the constraints allow a large number of tuples. Moreover, if the constraints are expressed in intention (e.g. by predicates), the memory space required for representing the dual instance may be greatly larger than one for representing the original instance. We can avoid building the dual by considering it in a virtual manner and by exploiting the alternative characterization of DBTP provided in Theorem 3. Indeed, in this characterization, we only need to manage the interactions between tuples of constraints. These interactions are directly expressed in the dual instance but can also be deduced on the fly by taking into account the intersection between tuples of different constraints. The second step can then be achieved by building and solving a max-closed instance P^o with a variable o_i (with domain $\{1, \ldots, e\}$) per constraint c_i of P and a constraint imposing $o_k < max(o_i, o_j)$ for all triples of constraints (c_i, c_j, c_k) s.t. $\exists t_i \in R(c_i)$, $t_j \in R(c_j)$ and $t_k, t'_k \in R(c_k)$, $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$, $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$, $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$, $t'_k[S(c_i) \cap S(c_k)] \neq t_i[S(c_i) \cap S(c_k)]$ and $t_j[S(c_j) \cap S(c_k)] \neq t_k[S(c_j) \cap S(c_k)]$.

Algorithm 1 implements this method. Given a CSP instance P, it returns *true* if P satisfies the DBTP property, false otherwise. Lines 1-8 are devoted to the construction of the max-closed instance P^o while Line 9 consists in solving this instance in polynomial time. Note that, if we consider the recognition method used in the proof of Lemma 2, checking whether the dual of P is BTP requires to build exactly the same max-closed instance as P^o, what ensures the validity of Algorithm 1.

We are interested here in the 7,272 benchmark instances from the CSP 2008 Compe-

tition[2]. These instances have binary or non-binary constraints which are represented in extension (by list of allowed tuples or list of forbidden tuples) or in intention (by predicate or global constraints). Most of the tractable classes we consider require that the constraints are expressed in extension by lists of allowed tuples in order to guarantee a polynomial recognition. Their recognition (including one of DBTP) may take from a few seconds to several hours (notably for instances having constraints in intention with large arity). So, for our experimentations, we exclude the instances containing constraints in intention with large arity (to ensure a reasonable runtime) or global constraints (because our CSP library does not take them into account yet). At the end, we have considered 2,800 instances for which we have checked the belonging to the classes $DBTP$ and BTP but also to some of the classes previously introduced, namely $TREE$, β-$ACYCLIC$, ZOA, CM, CCM, $IFUN$, MC, TR and BTW_k.

If we check the property directly on the original instances, the only instances that are detected as DBTP are those which are acyclic or β-acyclic. We find 8 binary instances with an acyclic constraint graph (e.g. all the instances of the `hanoi` family) and 23 non-binary instances with a β-acyclic constraint hypergraph.

Tables 1 and 2-3 provide respectively the list of the binary and non-binary instances which are detected as DBTP after having enforced the arc-consistency. They also indicate the value of some parameters of the instances and their tree-width[3] w or a range of values when the exact value is unknown (we recall that computing w is NP-hard). Table 1 presents the belonging of the original instances to the classes $DBTP$, MC, ZOA and CM (i.e. before enforcing arc-consistency) while Tables 2 and 3 do the same for the classes $DBTP$ and MC before the filtering and for the class MC after having enforced the arc-consistency.

The absence of instances for which DBTP holds thanks to relational properties is partially due to the presence of useless values in the domains and so to useless tuples in relations. To avoid this problem, a possible solution consists in simplifying the instances by enforcing some level of consistency (in the same spirit of *hidden tractable classes* introduced in [12]), namely here the arc-consistency. The choice of the arc-consistency is quite natural since most solvers exploit a level of consistency at least as powerful as the arc-consistency. By so doing, 362 instances belong trivially to DBTP or to any other relational or hybrid tractable classes since they are detected as inconsistent and so all the domains are empty (we assume that the filtering process is not stopped as soon as a domain becomes empty). We have found 105 instances which are arc-consistent and DBTP.

For binary instances, 37 instances have been detected as DBTP. We can note that all the instances of families `domino` and `hanoi` are DBTP after an AC filtering. Moreover, for the instances from families `hanoi` and the instances `graph12-w0` and `graph13-w0`, the DBTP property holds because these instances have an acyclic constraint graph. Finally, we have observed that, once the arc-consistency has been enforced, all the instances in Table 1 also belong to BTP, ZOA and TR while all of them except `graph12-w0` and `graph13-w0` are in CM, CC, $IFUN$ and MC. However, all the instances of the competition which are BTP are not necessarily DBTP. For example, the instance `fapp17-0300-10` is BTP, but not DBTP. Regarding the class BTW_k, the instances `large-*` can be assimilated to instances having an unbounded tree-width (even if formally, it is not the case since they are instances) since

[2]See http://www.cril.univ-artois.fr/CPAI08 for more details.

[3]We recall that this notion is extended to hypergraphs by considering the notion of primal graphs.

their tree-width is equal to their number of variables minus one.

Regarding non-binary instances, the instances of the family `mknap` are trivially DBTP and β-acyclic since each one contains a single constraint. Then, we can observe that more than 33% of instances from the `primes-*` families are DBTP. For ten of these instances, the DBTP property holds because they have a β-acyclic constraint hypergraph. It follows that the remaining instances are DBTP thanks to the features of their relations. So the `primes-*` families illustrate perfectly the fact that the class DBTP is hybrid. We note that all the DBTP instances are also triangular like in the binary case while none is incrementally functional. The instances of the `mknap` family belong to MC. It is the same for some instances of the `prime-*` families once the arc-consistency is enforced. Finally, the tree-width of the DBTP instances is not necessarily bounded due to the arbitrary arity of constraints. For example, the tree-width of the `mknap-1-6` is equal to its number of variables minus one.

Table 4 provides the list of families for which no instance is DBTP, what corresponds to 1,640 instances. The other instances which are not DBTP belong to families for which at least one instance is DBTP or has an unknown status. At the end, the DBTP instances only represent about 4% of the considered instances (17% if we also consider the instances which are arc-inconsistent and so trivially DBTP). However, on the one hand, this shows that this tractable class is not artificial, and on the other hand, their membership of DBTP will make it possible to explain the efficiency of their solving in the next part.

6.2 Links with the solving

Generally, the instances of a given tractable class are solved thanks to a specific algorithm which is dedicated to this particular tractable class. Here, our aim is not to solve the DBTP instances with a specific algorithm, but to exploit the DBTP property to explain why some instances are solved efficiently by classical algorithms like MAC [26] or RFL [23], on which most solvers of the state of the art rely. Of course, we focus our study on instances which are both DBTP and arc-consistent.

For the binary instances which are DBTP and arc-consistent, Theorem 5 states that MAC solves these instances in polynomial time. In practice, MAC turns to be very efficient since it solves all the instances listed in Table 1 in a backtrack-free manner.

Regarding the non-binary instances, the 54 DBTP instances of the `primes-*` families are also solved efficiently by MAC in a backtrack-free manner (except two which require a single backtrack). For 10 instances, the size of the largest intersection between the scopes of two constraints does not exceed one. So, Theorem 6 holds, what explains the solving efficiency we observe. However, this theorem concerns a particular case where the arc-consistency of the instance entails its pairwise-consistency. Other cases may exist depending on the features of relations. For 38 instances, enforcing the arc-consistency entails the pairwise-consistency at each step of the search and so MAC solves them in polynomial time as stated by Theorem 5. For 6 instances, enforcing the arc-consistency is unable to entail the pairwise-consistency, the membership of the class DBTP is not a sufficient reason to explain the efficient solving.

The `mps-sentoy` instance and the three first instances of the `mknap` family are solved efficiently in a backtrack-free manner. As they have a single constraint, Theorem 5 holds, what explains the efficient solving. In contrast, for the other instances of the `mknap` family or the instances of the `mps` family, MAC does not succeed in solving them efficiently. The explanation of this phenomenon is related to the assumption that the relations are expressed

Instance	n	e	d	w	$DBTP$	MC	ZOA	CM
domino-100-100	100	100	100	2	no	no	no	yes
domino-100-200	100	100	200	2	no	no	no	yes
domino-100-300	100	100	300	2	no	no	no	yes
domino-1000-100	1,000	1,000	100	2	no	no	no	yes
domino-1000-1000	1,000	1,000	1,000	2	no	no	no	yes
domino-1000-200	1,000	1,000	200	2	no	no	no	yes
domino-1000-300	1,000	1,000	300	2	no	no	no	yes
domino-1000-500	1,000	1,000	500	2	no	no	no	yes
domino-1000-800	1,000	1,000	800	2	no	no	no	yes
domino-2000-2000	2,000	2,000	2,000	2	no	no	no	yes
domino-300-100	300	300	100	2	no	no	no	yes
domino-300-200	300	300	200	2	no	no	no	yes
domino-300-300	300	300	300	2	no	no	no	yes
domino-3000-3000	3,000	3,000	3,000	2	no	no	no	yes
domino-500-100	500	500	100	2	no	no	no	yes
domino-500-200	500	500	200	2	no	no	no	yes
domino-500-300	500	500	300	2	no	no	no	yes
domino-500-500	500	500	500	2	no	no	no	yes
domino-5000-5000	5,000	5,000	5,000	2	no	no	no	yes
domino-800-100	800	800	100	2	no	no	no	yes
domino-800-200	800	800	200	2	no	no	no	yes
domino-800-300	800	800	300	2	no	no	no	yes
domino-800-500	800	800	500	2	no	no	no	yes
domino-800-800	800	800	800	2	no	no	no	yes
hanoi-3_ext	6	5	27	1	yes	no	no	no
hanoi-4_ext	14	13	81	1	yes	no	no	no
hanoi-5_ext	30	29	243	1	yes	no	no	no
hanoi-6_ext	62	61	729	1	yes	no	no	no
hanoi-7_ext	126	125	2,187	1	yes	no	no	no
large-80-sat_ext	80	3,160	80	79	no	no	no	no
large-84-sat_ext	84	3,486	84	83	no	no	no	no
large-88-sat_ext	88	3,828	88	87	no	no	no	no
large-92-sat_ext	92	4,186	92	91	no	no	no	no
large-96-sat_ext	96	4,560	96	95	no	no	no	no
mps-diamond	2	1	2	1	yes	yes	yes	yes
graph12-w0	680	340	44	1	yes	yes	no	no
graph13-w0	916	458	44	1	yes	yes	no	no

Table 1: List of binary instances which are detected as DBTP after having enforced the arc-consistency and for each instance, the values of some parameters, its tree-width w and its belonging or not to the classes $DBTP$, MC, ZOA and CM before enforcing the arc-consistency.

Instance	n	e	d	r	w	$DBTP$	MC	MC after AC
mknap-1-0	6	1	2	6	5	yes	yes	yes
mknap-1-2	15	1	2	15	14	yes	yes	yes
mknap-1-3	20	1	2	20	19	yes	yes	yes
mknap-1-4	28	1	2	28	27	yes	yes	yes
mknap-1-5	39	1	2	39	38	yes	yes	yes
mknap-1-6	50	1	2	50	49	yes	yes	yes
primes-10-20-2-1	100	20	28	3	2	yes	no	no
primes-10-20-3-1	100	20	28	4	3	yes	no	no
primes-10-40-2-1	100	40	28	3	2	no	no	no
primes-10-40-3-1	100	40	28	4	[3,9]	no	no	no
primes-10-60-2-1	100	60	28	3	[2,5]	no	no	no
primes-10-60-2-3	100	60	28	5	[4,21]	no	no	no
primes-10-60-2-5	100	60	28	7	[6,35]	no	no	no
primes-10-60-3-1	100	60	28	4	[3,20]	no	no	no
primes-10-80-2-1	100	80	28	3	[2,8]	no	no	yes
primes-10-80-2-3	100	80	28	5	[4,27]	no	no	no
primes-10-80-2-5	100	80	28	7	[6,42]	no	no	yes
primes-10-80-3-1	100	80	28	4	[3,27]	no	no	no
primes-10-80-3-3	100	80	28	6	[5,42]	no	no	yes
primes-15-20-2-1	100	20	46	3	2	yes	no	no
primes-15-20-3-1	100	20	46	4	3	yes	no	no
primes-15-40-2-1	100	40	46	3	2	no	no	no
primes-15-40-2-3	100	40	46	5	[4,10]	no	no	no
primes-15-40-3-1	100	40	46	4	[3,9]	no	no	no
primes-15-60-2-1	100	60	46	3	[2,5]	no	no	yes
primes-15-60-2-3	100	60	46	5	[4,21]	no	no	no
primes-15-60-3-1	100	60	46	4	[3,20]	no	no	no
primes-15-60-3-3	100	60	46	6	[5,34]	no	no	no
primes-15-80-2-1	100	80	46	3	[2,8]	no	no	yes
primes-15-80-2-3	100	80	46	5	[4,27]	no	no	no
primes-15-80-3-1	100	80	46	4	[3,27]	no	no	no
primes-15-80-3-3	100	80	46	6	[5,42]	no	no	yes
primes-20-20-2-1	100	20	70	3	2	yes	no	no
primes-20-20-3-1	100	20	70	4	3	yes	no	no

Table 2: List of non-binary instances which are detected as DBTP after having enforced the arc-consistency and for each instance, the values of some parameters, its tree-width w (or a range) and its belonging or not to the classes $DBTP$ and MC before enforcing the arc-consistency and to the class MC after having enforced the arc-consistency. A dash means that the computation cannot be achieved due to an expensive runtime.

Instance	n	e	d	r	w	$DBTP$	MC	MC after AC
primes-20-40-2-1	100	40	70	3	2	no	no	no
primes-20-40-3-1	100	40	70	4	[3,9]	no	no	no
primes-20-60-2-1	100	60	70	3	[2,5]	no	no	no
primes-20-60-2-3	100	60	70	5	[4,21]	no	no	no
primes-20-60-3-1	100	60	70	4	[3,20]	no	no	no
primes-20-80-2-1	100	80	70	3	[2,8]	no	no	yes
primes-20-80-2-3	100	80	70	5	[4,27]	no	no	no
primes-20-80-3-1	100	80	70	4	[3,27]	no	no	no
primes-25-20-2-1	100	20	96	3	2	yes	no	no
primes-25-20-3-1	100	20	96	4	3	yes	no	no
primes-25-40-2-1	100	40	96	3	2	no	no	no
primes-25-40-3-1	100	40	96	4	[3,9]	no	no	no
primes-25-60-2-1	100	60	96	3	[2,5]	no	no	no
primes-25-60-2-3	100	60	96	5	[4,21]	no	no	no
primes-25-60-3-1	100	60	96	4	[3,20]	no	no	no
primes-25-80-2-1	100	80	96	3	[2,8]	no	no	yes
primes-25-80-2-3	100	80	96	5	[4,27]	no	no	no
primes-25-80-3-1	100	80	96	4	[3,27]	no	no	no
primes-30-20-2-1	100	20	112	3	2	yes	no	no
primes-30-20-3-1	100	20	112	4	3	yes	no	no
primes-30-40-2-1	100	40	112	3	[2,2]	no	no	no
primes-30-40-3-1	100	40	112	4	[3,9]	no	no	no
primes-30-60-2-1	100	60	112	3	[2,5]	no	no	no
primes-30-60-3-1	100	60	112	4	[3,20]	no	no	no
primes-30-80-2-1	100	80	112	3	[2,8]	no	no	yes
primes-30-80-3-1	100	80	112	4	[3,27]	no	no	no
mps-sentoy	60	1	2	60	59	yes	no	-
mps-red-markshare1-1	280	11	2	130	129	yes	no	-
mps-red-markshare1	230	6	2	80	79	yes	no	-
mps-red-markshare2	270	7	2	90	89	yes	no	-
mps-red-blend2	2,944	196	2	2,659	2,658	yes	no	-
mps-red-est	146	4	2	74	73	yes	no	-
mps-red-markshare2-1	330	13	2	150	149	yes	no	-

Table 3: List of non-binary instances (Table 2 continued) which are detected as DBTP after having enforced the arc-consistency and for each instance, the values of some parameters, its tree-width w (or a range) and its belonging or not to the classes $DBTP$ and MC before enforcing the arc-consistency and to the class MC after having enforced the arc-consistency. A dash means that the computation cannot be achieved due to an expensive runtime.

binary		non binary
BH-4-4	frb50-23	aim-100
bqwh-15-106	frb56-25	aim-200
bqwh-18-141	frb59-26	aim-50
coloring	geom	dubois
composed-25-1-2	graphColoring/mug	pret
composed-25-1-25	graphColoring/sgb/book	pseudo/aim
composed-25-1-40	graphColoring/sgb/games	
composed-25-1-80	QCP-10	
composed-25-10-20	QCP-15	
composed-75-1-2	QCP-20	
composed-75-1-25	QWH-10	
composed-75-1-40	QWH-15	
composed-75-1-80	QWH-20	
ehi-85	rand-2-30-15-fcd	
ehi-90	rand-2-40-19	
fapp/fapp17	rand-2-40-19-fcd	
fapp/fapp18	Rand-2-50-23	
fapp/fapp19	rand-2-50-23-fcd	
fapp/fapp20	tightness0.1	
frb30-15	tightness0.2	
frb35-17	tightness0.35	
frb40-19	tightness0.5	
frb45-21		

Table 4: List of binary and non-binary families for which none instance is DBTP.

in extension and to the large value of the arity of the constraint. If, the violation of this assumption often has no consequence on the efficiency of MAC applied on DBTP instances (all the instances of Tables 2 and 3 have constraints defined by predicates), it is not the case here.

Finally, note that we have made the same observations when using RFL instead of MAC, what was foreseeable since Theorems 5 and 6 also hold for RFL.

7 Conclusion and future works

In this paper, we have studied a hybrid tractable class whose instances can be solved in polynomial time by MAC-like algorithms. We have then proved that it is incomparable with several known tractable classes (notably BTP) and that it captures both structural and relational tractable classes (namely β-acyclic CSPs and Triangular CSPs). We have also compared DBTP with the Directional Hyper-k-Consistency, which led us to present new results for BTP. Finally, we have analysed DBTP from a practical point of view. This study has shown that DBTP is not an artificial tractable class since several classical benchmarks among the ones used in the CSP 2008 Competition belong to DBTP. Moreover, for most of these instances, their membership of DBTP allows us to explain their ability to be efficiently solved by solvers of the state of the art based on algorithms like MAC or RFL.

A first extension consists in studying the link between DBTP and other tractable classes we have not mentioned in this paper. Another one consists in considering other properties and then in extending other tractable classes to non-binary CSPs using a similar approach, using the dual representation. One interesting candidate could be the *min-of-max extendable* property also introduced in [8].

Then, in the same spirit, we can also explore the possibility of defining new tractable classes, taking properties like BTP (or some others) and exploiting other encodings of non-binary CSPs.

Acknowledgements

This work was supported by the French National Research Agency under grant TUPLES (ANR-2010-BLAN-0210).

The authors would like to thank Martin C. Cooper for his useful comments.

References

[1] van Beek, P., Dechter, R.: On the minimality and decomposability of row-convex constraint networks. Journal of the ACM **42(3)**, 543–561 (1995)

[2] Beeri, C., Fagin, R., Maier, D., Yannakakis, M.: On the desirability of acyclic database schemes. Journal of the ACM **30**(3), 479–513 (1983)

[3] Bessière, C.: Constraint Propagation, chap. 3. Handbook of Constraint Programming. Elsevier (2006)

[4] Bessière, C., Stergiou, K., Walsh, T.: Domain filtering consistencies for non-binary constraints. Artificial Intelligence **172**, 800–822 (2008)

[5] Cohen, D., Cooper, M., Green, M., Marx, D.: On guaranteeing polynomially bounded search tree size. In: Proceedings of CP, pp. 160–171 (2011)

[6] Cohen, D.A.: A New Classs of Binary CSPs for which Arc-Constistency Is a Decision Procedure. In: Proceedings of CP 2003, pp. 807–811 (2003)

[7] Cooper, M., Cohen, D., Jeavons, P.: Characterising Tractable Constraints. Artificial Intelligence **65(2)**, 347–361 (1994)

[8] Cooper, M., Jeavons, P., Salamon, A.: Generalizing constraint satisfaction on trees: hybrid tractability and variable elimination. Artificial Intelligence **174**, 570–584 (2010)

[9] Duris, D.: Some characterizations of γ and β-acyclicity of hypergraphs. Information Processing Letters **112 (16)**, 617–620 (2012)

[10] El Mouelhi, A., Jégou, P., Terrioux, C., Zanuttini, B.: Some New Tractable Classes of CSPs and their Relations with Backtracking Algorithms. In: Proceedings of CP-AI-OR, pp. 61–76 (2013)

[11] El Mouelhi, A., Jégou, P., Terrioux, C.: A Hybrid Tractable Class for Non-Binary CSPs. In: Proceedings of ICTAI, pp. 947–954 (2013)

[12] El Mouelhi, A., Jégou, P., Terrioux, C.: Hidden Tractable Classes: from Theory to Practice. In: Proceedings of ICTAI, pp. 437–445 (2014)

[13] Fagin, R.: Degrees of Acyclicity for Hypergraphs and Relational Database Schemes. Journal of the ACM **30**(3), 514–550 (1983)

[14] Freuder, E.: A Sufficient Condition for Backtrack-Free Search. Journal of the ACM **29 (1)**, 24–32 (1982)

[15] Gavril, F.: Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. SIAM Journal on Computing **1 (2)**, 180–187 (1972)

[16] Golumbic, M.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980)

[17] Graham, M.H.: On the universal relation. Tech. rep., University of Toronto (1979)

[18] Janssen, P., Jégou, P., Nouguier, B., Vilarem, M.C.: A filtering process for general constraint satisfaction problems: achieving pairwise-consistency using an associated binary representation. In: Proceedings of IEEE Workshop on Tools for Artificial Intelligence, pp. 420–427 (1989)

[19] Jeavons, P., Cooper, M.: Tractable constraints on ordered domains. Artificial Intelligence **79(2)**, 327–339 (1995)

[20] Jégou, P.: Contribution à l'étude des problèmes de satisfaction de contraintes : Algorithmes de propagation et de résolution – Propagation de contraintes dans les réseaux dynamiques. Ph.D. thesis, Université des Sciences et Techniques du Languedoc (1991)

[21] Jégou, P.: Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In: Proceedings of AAAI, pp. 731–736 (1993)

[22] Jégou, P.: On the Consistency of General Constraint-Satisfaction Problems. In: Proceedings of AAAI, pp. 114–119 (1993)

[23] Nadel, B.: Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms, pp. 287–342. In *Search in Artificial Intelligence*. Springer-Verlag (1988)

[24] Robertson, N., Seymour, P.: Graph minors II: Algorithmic aspects of treewidth. Algorithms **7**, 309–322 (1986)

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

[26] Sabin, D., Freuder, E.: Contradicting Conventional Wisdom in Constraint Satisfaction. In: Proceedings of ECAI, pp. 125–129 (1994)

[27] Salamon, A., Jeavons, P.: Perfect Constraints Are Tractable. In: Proceedings of CP, pp. 524–528 (2008)