The Extendable-Triple Property: a new CSP Tractable Class beyond BTP

Philippe Jégou and **Cyril Terrioux**
Aix-Marseille Université, CNRS
LSIS UMR 7296
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{philippe.jegou, cyril.terrioux}@lsis.org

Abstract

Tractable classes constitute an important issue in Artificial Intelligence to define new islands of tractability for reasoning or problem solving. In the area of constraint networks, numerous tractable classes have been defined, and recently, the Broken Triangle Property (BTP (Cooper, Jeavons, and Salamon 2010)) has been shown as one of the most important of them, this class including several classes previously defined. In this paper, we propose a new class called ETP for *Extendable-Triple Property*, which generalizes BTP, by including it. Combined with the verification of the Strong-Path-Consistency, ETP is shown to be a new tractable class. Moreover, this class inherits some desirable properties of BTP including the fact that the instances of this class can be solved thanks to usual algorithms (such as MAC or RFL) used in most solvers. We give the theoretical material about this new class and we present an experimental study which shows that from a practical viewpoint, it seems more usable in practice than BTP.

Introduction

Constraint Satisfaction Problems (CSPs (Rossi, van Beek, and Walsh 2006)) constitute an important formalism of Artificial Intelligence (AI) for expressing and efficiently solving a wide range of practical problems. A constraint network (or CSP, abusing words) consists of a set X of variables, each of which must be assigned a value in its associated (finite) domain, so that these assignments together satisfy a finite set C of constraints.

Deciding whether a given CSP has a solution is an NP-complete problem. Hence classical approaches to this problem are based on backtracking algorithms, whose worst-case time complexity is generally in $O(e.d^n)$ with n the number of variables, e the number of constraints and d the size of the largest domain. To increase their efficiency, such algorithms also rely on filtering techniques during search (among other techniques, such as variable ordering heuristics or constraint learning). With the help of such techniques, despite their theoretical time complexity, algorithms such as Forward Checking (Haralick and Elliot 1980), RFL (for Real Full Look-ahead (Nadel 1988)) or MAC (for Maintaining Arc Consistency (Sabin and Freuder 1994)) turn out to be very efficient in practice on a wide range of practical problems. Although these methods are effective in practice, their time complexity is exponential. To ensure better computation times, many studies have been developed to highlight tractable classes, that is to say sets of instances that can be solved in polynomial time. Unfortunately, these works have often been of a theoretical interest only, without allowing to improve solving capabilities in practice. So, these classes are not in fact used in practice. This is mainly due to the fact that these classes are often artificial in the sense that they define instances that do not appear in practice. In addition, it is quite difficult to integrate the management of these classes in the solvers of the state of the art. Indeed, tractable classes are generally handled by ad hoc algorithms. A first algorithm must decide if an instance belongs to a given tractable class and then, if so, a second algorithm solves the instance. The implementation of these algorithms in solvers thus leads to considerable extra cost, even if both algorithms are very efficient (e.g. in linear time). Hence, in our opinion, in order to be exploited during the solving, the tractable classes must be implicitly handled by classical solvers (i.e. that these solvers are able to solve the instances of these classes in polynomial time without any additional processing). For example, this is the case of the BTP class (Cooper, Jeavons, and Salamon 2010) whose instances can be solved in polynomial time by MAC without any additional processing. Such tractable classes can then be useful to explain the good efficiency of solvers on some benchmarks.

In this paper, we study a possible extension of BTP. The primary motivation of this study is related to the interest of BTP, both from a theoretical viewpoint (this hybrid tractable class includes well known tractable classes of the literature) and from a practical viewpoint (this class has very useful properties including the possibility of being implicitly exploited by the solvers of the state of the art). In addition, we have observed that BTP can be extended by relaxing some of the restrictions it imposes. Indeed, the tractability of the solving of the instances verifying BTP is based on a property which appears to be very strong. This property is induced by the conditions allowing to define BTP. So, we show that it is possible to relax this property while preserving the tractability. This leads to define a weaker property called ETP for Extendable-Triple Property which is recognizable in polynomial time. As for BTP,

this property is definable by the notion of *broken triangles*. However while BTP is defined by the existence of an ordering prohibiting the existence of any broken triangles, ETP tolerates some broken triangles. From a theoretical viewpoint, this approach makes it possible to generalize the BTP property by defining a new class that includes it strictly. It also allows us to offer a class whose instances can potentially appear more easily in practice. Combined with the verification of the Strong-Path-Consistency (Freuder 1982), ETP is shown to be a new tractable class. The fact that the ETP property generalizes the BTP property is experimentally verified in this paper showing that it occurs more frequently than BTP in the benchmarks used by the community. In addition, we show that it can sometimes help to explain why the solving of some benchmarks by algorithms such as MAC or RFL is really efficient even for instances that do not belong to BTP. Finally, we show that this new tractable class turns out to be more exploitable in practice, particularly by approaches based on the notion of *Backdoor* (Williams, Gomes, and Selman 2003).

The paper is organized as follows. The next section introduces notations and recalls some notions about tractable classes, and more precisely, about BTP. The third section presents the theoretical basis of this extension, while the fourth section introduces the property ETP and the associated tractable class. Then we illustrate its practical interest by an experimental study before concluding.

Background

Formally, a *constraint satisfaction problem* also called *constraint network* is a triple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is a set of n variables, $D = (D_{x_1}, \ldots, D_{x_n})$ is a list of finite domains of values, one per variable, and $C = \{c_1, \ldots, c_e\}$ is a finite set of e constraints. Each constraint c_i is a pair $(S(c_i), R(c_i))$, where $S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ is the *scope* of c_i, and $R(c_i) \subseteq D_{x_{i_1}} \times \cdots \times D_{x_{i_k}}$ is its *compatibility relation*. The *arity* of c_i is $|S(c_i)|$. In this paper, we only deal with the case of binary CSPs, that is CSPs for which all the constraints are of arity 2. Hence, we will denote by c_{ij} the constraints involving x_i and x_j. The structure of a constraint network is represented by a graph, called the *constraint graph*, whose vertices correspond to variables and edges to the constraint scopes. An assignment on a subset of X is said to be *consistent* if it does not violate any constraint. Testing whether a CSP has a *solution* (i.e. a consistent assignment on all the variables) is known to be NP-complete. So, many works have been realized to make the solving of instances more efficient in practice, by using optimized backtracking algorithms, heuristics, constraint learning, non-chronological backtracking, filtering techniques based on constraint propagation, etc. The worst-case time complexity for these approaches is naturally exponential, at least in $O(e.d^n)$ where n is the number of variables, d the maximum size of domains and e the number of constraints. Despite this exponential complexity, the solvers of the state of the art have shown in many cases their practical efficiency.

Although the problem CSP is NP-complete, there exist classes of instances that can be recognized[1] and solved 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 properties to define tractable classes. The first one concerns the structural properties of the constraint network. For example, tree-structured (i.e. acyclic) binary CSPs can be solved in linear time (Freuder 1982). 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" (Cooper, Cohen, and Jeavons 1994). More recently, a kind of tractable classes has been proposed such as the BTP class (Cooper, Jeavons, and Salamon 2010). Their interest is that they are able to take into account both language and structure restrictions. They are thus sometimes called "*hybrid classes*". We recall the BTP property:

Definition 1 (Broken Triangle Property) *A binary 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. $x_i < x_j < x_k$, if $(v_i, v_j) \in R(c_{ij})$, $(v_i, v_k) \in R(c_{ik})$ 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, (v_i, v_j, v_k, v'_k) is called a* broken triangle on x_k w.r.t. x_i and x_j. *If there exists at least one broken triangle on x_k w.r.t. x_i and x_j, (x_i, x_j, x_k) is called a* broken triple on x_k w.r.t. x_i and x_j. *Let BTP be the set of the instances for which BTP holds w.r.t. some variable ordering.*

The BTP property is related to the compatibility between the values of domains which can be graphically visualized (Figure 1) on the microstructure graph[2]. E.g. in Figure 1 (a), there is a broken triangle on x_3 with respect to the variables x_1 and x_2 since we have $(v_1, v'_3) \notin R(c_{13})$ and $(v_2, v_3) \notin R(c_{23})$ while $(v_1, v_2) \in R(c_{12})$, $(v_1, v_3) \in R(c_{13})$ and $(v_2, v'_3) \in R(c_{23})$ hold. So (x_1, x_2, x_3) is a broken triple on x_3 w.r.t. x_1 and x_2. In contrast, in Figure 1 (b), if one of the two dashed edges (that is binary tuples) appears in the microstructure, the BTP property holds independently from the ordering.

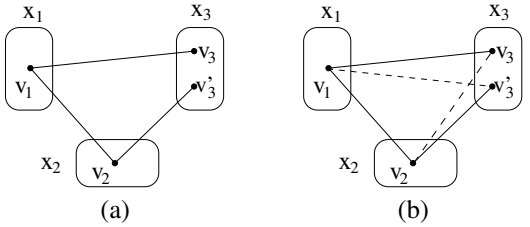

Figure 1: A non-BTP pattern (a) and a BTP one (b) w.r.t. the order $x_1 < x_2 < x_3$.

[1]Here, we consider the definition proposed in (Gottlob, Leone, and Scarcello 2000).

[2]The micro-structure (Jégou 1993) of a binary CSP $P = (X, D, C)$ is the undirected graph $\mu(P) = (V, E)$ where $V = \{(x_i, v_i) : x_i \in X, v_i \in D_{x_i}\}$ and $E = \{\{(x_i, v_i), (x_j, v_j)\} : i \neq j, c_{ij} \notin C$ or $(v_i, v_j) \in R(c_{ij})\}$.

The time complexity to recognize an instance of the class BTP (i.e. to show that there exists a variable ordering satisfying the BTP property) is $O(n.e.d^3)$. Among the most important properties of BTP, first of all, we note that it includes structural tractable classes (e.g. acyclic binary CSPs), but also tractable classes defined by language restrictions (e.g. RRM (Cooper, Jeavons, and Salamon 2010)). Moreover, this class is conservative, i.e. it is closed under domain restrictions. This is particularly interesting since it allows Cooper et al. to show that MAC (and thus also RFL) can solve the instances of BTP in polynomial time, precisely in $O(n.e.d^3)$, whatever the instantiation ordering of the variables during search. For these different reasons, BTP seems to be the most interesting tractable class defined these last years.

How can we generalize BTP ?

The fact that the class BTP is a very interesting class thanks to its properties encourages us try to extend it. For this, it would be desirable to retain its most useful properties, including the fact that it is treatable in polynomial time by algorithms such as MAC, without having to recognize its instances. One possible way is then to relax the conditions imposed in its definition which, ultimately seem to be excessive. We develop this point below, but this requires to remind of some of its associated properties. The first one is Lemma 2.4 from (Cooper, Jeavons, and Salamon 2010):

Lemma 1 ((Cooper, Jeavons, and Salamon 2010)) *A binary CSP instance satisfies the broken-triangle property with respect to the variable ordering $<$ if, and only if, $\forall i < j < k$ and $\forall (u, v) \in R(c_{ij})$:*

$$(R(c_{ik})[u] \subseteq R(c_{jk})[v]) \text{ or } (R(c_{jk})[v] \subseteq R(c_{ik})[u])$$

with $R(c_{ij})[a] = \{b \in D_{x_j} : (a, b) \in R(c_{ij})\}$.

From this lemma, it is easy to prove the next property which shows that an instance which belongs to BTP is solvable in polynomial time. This lemma comes from the proof of Theorem 3.1 in (Cooper, Jeavons, and Salamon 2010).

Lemma 2 ((Cooper, Jeavons, and Salamon 2010)) *If a binary CSP instance satisfies the broken-triangle property with respect to the variable ordering $<$ and if $(u_1, u_2, \dots u_{k-1})$ is a consistent assignment of $(x_1, x_2, \dots x_{k-1})$, then the set $\{R(c_{ik})[u_i] : i < k\}$ is totally ordered by subset inclusion.*

The proof that the instances which satisfy BTP can be solved in polynomial time is based on the fact that from Lemma 2, one can deduce that the set $\{R(c_{ik})[u_i] : i < k\}$ has a minimal element associated to a rank i_0 such that $i_0 < k$ and such that $R(c_{i_0k})[u_{i_0}] = \bigcap_{i<k} R(c_{ik})[u_i]$. A graphical representation of Lemma 2 is given in Figure 2. In this figure, a value $u_i \in D_{x_i}$ is connected by an edge to a subset $R(c_{ik})[u_i]$ of D_{x_k} materialized by an ellipse (i.e. u_i is compatible with all its values).

In Figure 2, we can see that given a consistent assignment $(u_1, u_2, \dots u_{k-1})$ of variables $(x_1, x_2, \dots x_{k-1})$, restricted in Figure 2 to the consistent assignment $(u_1, \dots u_i, \dots u_{i_0}, \dots u_{k-1})$ of the variables

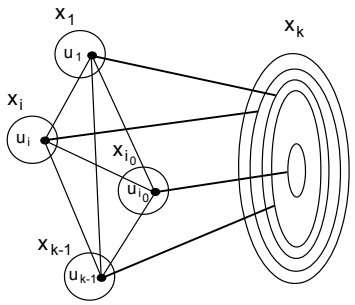

Figure 2: Graphical explanation of the property on sub-domain inclusion showing the tractability of BTP instances.

$(x_1, \dots x_i, \dots x_{i_0}, \dots x_{k-1})$, it is possible to extend this assignment to x_k preserving consistency with a value belonging to $R(c_{i_0k})[u_{i_0}] \subseteq D_{x_k}$. This is the case if $R(c_{i_0k})[u_{i_0}] \neq \emptyset$, which is valid assuming that the instance satisfies the arc-consistency. It is noted here that BTP leads to a sufficient condition to prove the existence of a polynomial time algorithm to solve the instances, but this condition does not seem fundamentally necessary. Indeed, it might be satisfied if there exists at least one value which appears in $\bigcap_{i<k} R(c_{ik})[u_i]$ independently of the existence of a minimal element $R(c_{i_0k})[u_{i_0}]$. A graphical representation is given in Figure 3 where we can see that every consistent assignment of $(x_1, x_2, \dots x_{k-1})$ can be extended by a value $u_k \in \bigcap_{i<k} R(c_{ik})[u_i]$.

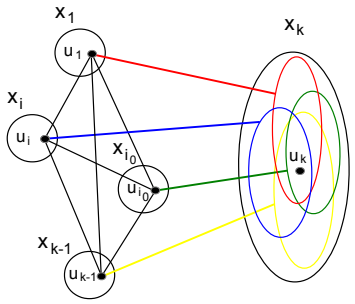

Figure 3: Graphical explanation of a weaker condition on sub-domain intersections.

We refer to this relaxation of BTP by the property we call *ETP* for *Extendable-Triple Property* and which is described in the next section.

ETP: Definition and Properties

The relaxation of the conditions of BTP we want while preserving the existence of a polynomial time algorithm for solving instances can be achieved using a surprising way related to a condition on broken triples which can exist in the subsets of four variables:

Definition 2 (Extendable-Triple Property (ETP)) *A binary CSP instance P satisfies the Extendable-Triple Property (ETP) with respect to the variable ordering $<$ if, and only*

if, for all subset of four variables (x_i, x_j, x_k, x_l) such that $i < j < k < l$, there is at most one broken triple on x_l among (x_i, x_j, x_l), (x_i, x_k, x_l) and (x_j, x_k, x_l).

Note that a binary CSP can satisfy the ETP property while it contains two broken triples among (x_i, x_j, x_k, x_l), one on x_k, and another one on x_l, while none is possible with BTP. An immediate consequence of this definition is given by the next theorem where BTP and ETP denote respectively the associated classes of instances.

Theorem 1 $BTP \subsetneq ETP$

To analyze the tractability of ETP, we have now to show that the instances of this class can be recognized in polynomial time:

Theorem 2 *Given a binary CSP instance $P = (X, D, C)$, there is a polynomial time algorithm to find a variable ordering $<$ such that P satisfies ETP with respect to $<$, or to determine that no such ordering exists.*

Proof: Like in the proof of Theorem 3.2 in (Cooper, Jeavons, and Salamon 2010), we define a CSP instance P_o which is consistent if and only if a possible ordering exists. More precisely, in this instance, we consider a variable o_i with domain $\{1, \ldots, n\}$ per variable x_i of X. o_i represents the position of the variable x_i in the ordering. Regarding BTP, Cooper et al. add a constraint $o_k < max(o_i, o_j)$ when there exists a broken triangle on x_k with respect to x_i and x_j. In similar way, we add a constraint involving $\{o_i, o_j, o_k, o_l\}$ and imposing the condition $o_l < max(o_i, o_j, o_k)$ for each quadruple of variables (x_i, x_j, x_k, x_l) such that there are at least two broken triples on x_l among (x_i, x_j, x_l), (x_i, x_k, x_l) and (x_j, x_k, x_l).

If P_o has a solution, then for each quadruple of variables (x_i, x_j, x_k, x_l), we have at most one broken triple among (x_i, x_j, x_l), (x_i, x_k, x_l) and (x_j, x_k, x_l). Indeed, if no constraint $o_l < max(o_i, o_j, o_k)$ exists, there is no broken triple. Otherwise, as the constraint is satisfied, x_l cannot appear after both x_i, x_j and x_k in the ordering. Without loss of generality, let us assume that x_l appears before x_i in the ordering (i.e. $o_l < o_i$). Then only the triple (x_j, x_k, x_l) can be broken. So, if P_o has a solution, we have an ordering satisfying the ETP property. Conversely, let us consider an ordering satisfying the ETP property and assume that P_o has no solution. It means that at least one constraint $o_l < max(o_i, o_j, o_k)$ for a given quadruple (x_i, x_j, x_k, x_l) is violated. So there are at least two broken triples on x_l among (x_i, x_j, x_l), (x_i, x_k, x_l) and (x_j, x_k, x_l) w.r.t. the considered ordering, which is impossible since this ordering satisfies the ETP property. Hence P_o has a solution if and only if P admits an ordering satisfying the ETP property.

We now consider the cost of building and solving the instance P_o. Finding all the broken triples can be achieved in $O(n^3.d^4)$ while defining the constraints $o_l < max(o_i, o_j, o_k)$ can be done in $O(n^4)$. So P_o can be computed in $O(n^3.d^4)$. Moreover, all the constraints of P_o are *max-closed* (Jeavons and Cooper 1995). Indeed, any constraint $o_l < max(o_i, o_j, o_k)$ of P_o is such that if we have $v_l < max(v_i, v_j, v_k)$ and $v'_l < max(v'_i, v'_j, v'_k)$, we have necessary $max(v_l, v'_l) < max(max(v_i, v'_i), max(v_j, v'_j), max(v_k, v'_k))$. As max-closed constraints can be solved easily by establishing generalized arc-consistency, the instance can be built and solved in polynomial time. □

We analyze now the solving of the instances of the class ETP.

Lemma 3 *A binary CSP instance P satisfies ETP with respect to the variable ordering $<$ if, and only if, for all subset of four variables x_i, x_j, x_k and x_l such that $i, j, k < l$, for all consistent assignment (u, v, w) of (x_i, x_j, x_k), at least two of the next properties are satisfied:*

1. $(R(c_{il})[u] \subseteq R(c_{jl})[v])$ *or* $(R(c_{jl})[v] \subseteq R(c_{il})[u])$
2. $(R(c_{il})[u] \subseteq R(c_{kl})[w])$ *or* $(R(c_{kl})[w] \subseteq R(c_{il})[u])$
3. $(R(c_{jl})[v] \subseteq R(c_{kl})[w])$ *or* $(R(c_{kl})[w] \subseteq R(c_{jl})[v])$

Proof: By hypothesis, there is no broken triple on at least two of the triples of variables (x_i, x_j, x_l), (x_i, x_k, x_l) and (x_j, x_k, x_l), which corresponds to a local satisfaction of BTP on at least two of these triples of variables. By Lemma 1, at least two of these properties are satisfied. □

Based on this lemma, we can propose a second lemma:

Lemma 4 *Given a binary CSP instance P satisfying ETP with respect to the variable ordering $<$, then, for all subset of four variables x_i, x_j, x_k and x_l such that $i, j, k < l$, for all consistent assignment (u, v, w) of (x_i, x_j, x_k), we have at least one of the next four possibilities (or similar properties obtained by permutation of i, j and k):*

4.1. $R(c_{il})[u] \subseteq R(c_{jl})[v] \cap R(c_{kl})[w]$
4.2. $R(c_{kl})[w] \subseteq R(c_{il})[u] \subseteq R(c_{jl})[v]$
4.3. $R(c_{jl})[v] \subseteq R(c_{il})[u] \subseteq R(c_{kl})[w]$
4.4. $R(c_{jl})[v] \cap R(c_{kl})[w] \subseteq R(c_{il})[u]$

Proof: By Lemma 3, we know that at least two of the three properties (1), (2) and (3) are satisfied. We must analyze three possible cases: (1) and (2) are satisfied, (1) and (3) are satisfied, and (2) and (3) are satisfied. Without loss of generality, since we have no hypothesis on the ordering between x_i, x_j and x_k, it is sufficient to consider only one of the three cases, the two other ones being equivalent. So, we consider the case such that (1) and (2) are satisfied. In this case, we have four possibilities:

4.1. $(R(c_{il})[u] \subseteq R(c_{jl})[v])$ and $(R(c_{il})[u] \subseteq R(c_{kl})[w])$. In this case $R(c_{il})[u] \subseteq R(c_{jl})[v] \cap R(c_{kl})[w]$.

4.2. $(R(c_{il})[u] \subseteq R(c_{jl})[v])$ and $(R(c_{kl})[w] \subseteq R(c_{il})[u])$. In this case $R(c_{kl})[w] \subseteq R(c_{il})[u] \subseteq R(c_{jl})[v]$.

4.3. $(R(c_{jl})[v] \subseteq R(c_{il})[u])$ and $(R(c_{il})[u] \subseteq R(c_{kl})[w])$. In this case $R(c_{jl})[v] \subseteq R(c_{il})[u] \subseteq R(c_{kl})[w]$.

4.4. $(R(c_{jl})[v] \subseteq R(c_{il})[u])$ and $(R(c_{kl})[w] \subseteq R(c_{il})[u])$ In this case $R(c_{jl})[v] \cap R(c_{kl})[w] \subseteq R(c_{il})[u]$. □

While cases 4.1, 4.2 and 4.3 allow to extend any consistent assignment (u, v, w) of three variables x_i, x_j and x_k

to a fourth variable x_l (assuming that the considered instance satisfies arc-consistency), the case 4.4 does not ensure a consistent assignment of x_l. Indeed, such an extension is possible only if $R(c_{jl})[v] \cap R(c_{kl})[w] \neq \emptyset$. A possible way to ensure that this property holds is to add to ETP an additional condition related to the satisfaction of path-consistency (Montanari 1974). If such a consistency holds, necessarily, the tuple (v, w) will be supported a least by one value in D_{x_l}. And thus, this value will necessarily be included in $R(c_{jl})[v] \cap R(c_{kl})[w]$, ensuring that this intersection will not be empty. So, to ensure the tractability of the class ETP, we consider this additional condition. We will consider instances satisfying arc and path-consistency, that is *Strong-Path-Consistency* (Freuder 1982), which is generally denoted SPC.

Theorem 3 *A binary CSP instance P satisfying SPC and ETP with respect to the variable ordering $<$ is consistent and a solution can be found in polynomial time.*

Proof: We consider an ordering for variable assignments corresponding to the ordering $<$. As the instance satisfies SPC, it satisfies arc-consistency and thus, no domain is empty and each value has a support in each domain. Moreover, as the instance satisfies SPC, it also satisfies path-consistency, and thus we assume that we have a consistent assignment on the three first variables. Now, and more generally, suppose that we have a consistent assignment $(u_1, u_2, \dots u_{l-1}, u_l)$ for the l first variables $x_1, x_2, \dots x_{l-1}, x_l$ in the ordering, with $3 \leq l < n$. We show that this assignment can be consistently extended to the variable x_{l+1}. To show this, we must prove that $\cap_{1 \leq i \leq l} R(c_{il+1})[u_i] \neq \emptyset$, that is there is at least one value in the domain of x_{l+1} which is compatible with the assignment $(u_1, u_2, \dots u_{l-1}, u_l)$.

We first prove this for $l = 3$. Consider the consistent assignment (u_i, u_j, u_k) on the three first variables x_i, x_j and x_k. Consider a fourth variable x_{l+1} appearing later in the ordering. By Lemma 4, we have four possibilities:

4.1. $R(c_{il+1})[u_i] \subseteq R(c_{jl+1})[u_j] \cap R(c_{kl+1})[u_k]$. Since the instance satisfies arc-consistency, $R(c_{il+1})[u_i] \neq \emptyset$ and then there exists at least one value $u_{l+1} \in R(c_{il+1})[u_i]$ and this value is compatible with u_j and u_k.

4.2. $R(c_{kl+1})[u_k] \subseteq R(c_{il+1})[u_i] \subseteq R(c_{jl+1})[u_j]$. Since the instance satisfies arc-consistency, $R(c_{kl+1})[u_k] \neq \emptyset$. Thus, there is at least one value $u_{l+1} \in R(c_{kl+1})[u_k]$ and this value is compatible with u_i and u_j.

4.3. $R(c_{jl+1})[u_j] \subseteq R(c_{il+1})[u_i] \subseteq R(c_{kl+1})[u_k]$. Since the instance satisfies arc-consistency, $R(c_{jl+1})[u_j] \neq \emptyset$. Thus, there is at least one value $u_{l+1} \in R(c_{jl+1})[u_j]$ and this value is compatible with u_i and u_k.

4.4. $R(c_{jl+1})[u_j] \cap R(c_{kl+1})[u_k] \subseteq R(c_{il+1})[u_i]$. Since the instance satisfies path-consistency, the pair (u_j, u_k) has at least one support u_{l+1} which belongs to the domain of x_{l+1}, and then, necessarily, $u_{l+1} \in R(c_{jl+1})[u_j] \cap R(c_{kl+1})[u_k]$. Moreover, this value u_{l+1} is compatible with u_i. Thus, the consistent assignment (u_i, u_j, u_k) can be extended by the value u_{l+1} to the variable x_{l+1}.

Note that this proof holds for all variables u_i, u_j, u_k and x_{l+1} such that x_{l+1} appears later in the ordering $<$, not only for the variables x_1, x_2, x_3 and x_4.

Now, we prove the property for l with $3 < l < n$. That is, we show that a consistent assignment $(u_1, u_2, \dots u_{l-1}, u_l)$ can be extended to a $(l+1)^{th}$ variable, which is equivalent to prove that $\cap_{1 \leq i \leq l} R(c_{il+1})[u_i] \neq \emptyset$. As induction hypothesis, we assume that every consistent assignment on $l-1$ variables can be extended to a l^{th} variable, which appears later in the considered ordering $<$. So, if $(u_{i_1}, u_{i_2}, \dots u_{i_{l-1}})$ is a partial (consistent) assignment included in $(u_1, u_2, \dots u_{l-1}, u_l)$, with $i_1, i_2, \dots i_{l-1} < m$ with $l+1 \leq m \leq n$, we have $\cap_{1 \leq j \leq l-1} R(c_{i_j m})[u_{i_j}] \neq \emptyset$. In particular, we have $\cap_{1 \leq j \leq l-1} R(c_{i_j l+1})[u_{i_j}] \neq \emptyset$.

Consider a consistent assignment $(u_1, u_2, \dots u_{l-1}, u_l)$ on the l first variables. By the induction hypothesis, each partial assignment $(u_{i_1}, u_{i_2}, \dots u_{i_{l-1}})$ of $(u_1, u_2, \dots u_{l-1}, u_l)$ can be extended to a consistent assignment $(u_{i_1}, u_{i_2}, \dots u_{i_{l-1}}, u_{l+1})$. In other words, we have $\cap_{1 \leq j \leq l-1} R(c_{i_j l+1})[u_{i_j}] \neq \emptyset$, and in particular, we have $\cap_{1 \leq i \leq l-1} R(c_{il+1})[u_i] \neq \emptyset$. We show now that we have $\cap_{1 \leq i \leq l-1} R(c_{il+1})[u_i] \cap R(c_{ll+1})[u_l] \neq \emptyset$.

Consider the assignment (u_j, u_k, u_l) with $1 \leq j, k < l$. As above, by Lemma 4, we have four kinds of possibilities for the relations between the three sets $R(c_{jl+1})[u_j]$, $R(c_{kl+1})[u_k]$ and $R(c_{ll+1})[u_l]$. We enumerate them, knowing that i_1, i_2 and i_3 which are pairwise different, can be j, k or l:

4.1. $R(c_{i_1 l+1})[u_{i_1}] \subseteq R(c_{i_2 l+1})[u_{i_2}] \cap R(c_{i_3 l+1})[u_{i_3}]$.

First, as the instance satisfies arc-consistency, $R(c_{i_1 l+1})[u_{i_1}] \neq \emptyset$. We have two possibilities since either $l \neq i_1$, or $l = i_1$. We analyze these two cases.

If $l \neq i_1$, we have $R(c_{jl+1})[u_j] \subseteq R(c_{ll+1})[u_l]$, or $R(c_{kl+1})[u_k] \subseteq R(c_{ll+1})[u_l]$, and thus $\cap_{1 \leq i \leq l-1} R(c_{il+1})[u_i] \subseteq R(c_{ll+1})[u_l]$. Then, we have $\cap_{1 \leq i \leq l-1} R(c_{il+1})[u_i] \cap R(c_{ll+1})[u_l] \neq \emptyset$, and also $\cap_{1 \leq i \leq l} R(c_{il+1})[u_i] \neq \emptyset$.

Otherwise, we have $l = i_1$ (and $j = i_2$ or $j = i_3$), that is $R(c_{ll+1})[u_l] \subseteq R(c_{jl+1})[u_j] \cap R(c_{kl+1})[u_k]$. By induction hypothesis, we know that the partial assignment including u_l but without u_j, that is $(u_1, u_2, \dots u_{j-1}, u_{j+1}, \dots u_{l-1}, u_l)$ can be extended to the variable x_{l+1}.
This is equivalent to
$(\cap_{1 \leq i \leq j-1} R(c_{il+1})[u_i]) \cap (\cap_{j+1 \leq i \leq l} R(c_{il+1})[u_i]) \neq \emptyset$.
But as $R(c_{ll+1})[u_l] \subseteq R(c_{jl+1})[u_j]$, we have
$(\cap_{1 \leq i \leq j-1} R(c_{il+1})[u_i]) \cap (\cap_{j+1 \leq i \leq l} R(c_{il+1})[u_i]) \subseteq R(c_{jl+1})[u_j]$
and thus
$(\cap_{1 \leq i \leq j-1} R(c_{il+1})[u_i]) \cap R(c_{jl+1})[u_j] \cap$

$(\cap_{j+1\leq i\leq l} R(c_{il+1})[u_i]) \neq \emptyset$, which is equivalent to $\cap_{1\leq i\leq l} R(c_{il+1})[u_i] \neq \emptyset$.

Finally in every cases we have $\cap_{1\leq i\leq l} R(c_{il+1})[u_i] \neq \emptyset$.

4.2. (and 4.3.) The cases 4.2 and 4.3 can be analyzed simultaneously since they correspond to one sole case that can be denoted:

$$R(c_{i_3 l+1})[u_{i_3}] \subseteq R(c_{i_1 l+1})[u_{i_1}] \subseteq R(c_{i_2 l+1})[u_{i_2}].$$

First, as the instance satisfies arc-consistency, $R(c_{i_3 l+1})[u_{i_3}] \neq \emptyset$. The different possibilities can be reduced to two cases, either $l \neq i_3$, or $l = i_3$. We analyze these two cases.

If $l \neq i_3$, assume that $j = i_3$. So, we have $R(c_{jl+1})[u_j] \subseteq R(c_{ll+1})[u_l]$, and thus $\cap_{1\leq i\leq l-1} R(c_{il+1})[u_i] \subseteq R(c_{ll+1})[u_l]$. So we have $\cap_{1\leq i\leq l-1} R(c_{il+1})[u_i] \cap R(c_{ll+1})[u_l] \neq \emptyset$, which is equivalent to $\cap_{1\leq i\leq l} R(c_{il+1})[u_i] \neq \emptyset$. Note that if $k = i_3$, we apply the same reasoning, but interchanging j and k.

Now, if $l = i_3$, assume that $j = i_1$ and $k = i_2$. Thus, we have $R(c_{ll+1})[u_l] \subseteq R(c_{jl+1})[u_j] \subseteq R(c_{kl+1})[u_k]$, and then also $R(c_{ll+1})[u_l] \subseteq R(c_{jl+1})[u_j] \cap R(c_{kl+1})[u_k]$. So, by the same reasoning than in the case 4.1 for $l = i_1$, we have $\cap_{1\leq i\leq l} R(c_{il+1})[u_i] \neq \emptyset$. Note that if $j = i_2$ and $k = i_1$, we apply the same reasoning, but interchanging j and k.

Finally, in every cases, we have $\cap_{1\leq i\leq l} R(c_{il+1})[u_i] \neq \emptyset$.

4.4. $R(c_{i_2 l+1})[u_{i_2}] \cap R(c_{i_3 l+1})[u_{i_3}] \subseteq R(c_{i_1 l+1})[u_{i_1}]$.

First, as the instance satisfies arc-consistency, $R(c_{i_2 l+1})[u_{i_2}] \neq \emptyset$ and $R(c_{i_3 l+1})[u_{i_3}] \neq \emptyset$. The different possibilities can be reduced to two cases, either $l = i_1$ or $l \neq i_1$. We analyze these two cases.

If $l = i_1$, we have $\cap_{1\leq i\leq l-1} R(c_{il+1})[u_i] \subseteq R(c_{ll+1})[u_l]$ and thus $\cap_{1\leq i\leq l-1} R(c_{il+1})[u_i] \cap R(c_{ll+1})[u_l] \neq \emptyset$, which is equivalent to $\cap_{1\leq i\leq l} R(c_{il+1})[u_i] \neq \emptyset$.

Now, if $l = i_2$ (it is the same proof for $l = i_3$). Assume that $j = i_1$ and $k = i_3$ (it is the same proof for $j = i_3$ and $k = i_1$, interchanging j and k). In this case, we have $R(c_{ll+1})[u_l] \cap R(c_{kl+1})[u_k] \subseteq R(c_{jl+1})[u_j]$. So, by the same reasoning than in the case 4.1 for $l = i_1$, we have $\cap_{1\leq i\leq l} R(c_{il+1})[u_i] \neq \emptyset$.

So, in every cases, we have $\cap_{1\leq i\leq l} R(c_{il+1})[u_i] \neq \emptyset$. Thus, every consistent assignment $(u_1, u_2, \ldots u_{l-1}, u_l)$ on $(x_1, x_2, \ldots x_{l-1}, x_l)$ can be extended to a $(l+1)^{th}$ variable, for all l with $3 < l < n$. And more generally, we have shown that every consistent assignment on l variables, not necessarily consecutive in the ordering (as are the l first variables), can be extended to a consistent assignment for every $(l+1)^{th}$ variable which appears after these l variables in the ordering $<$ associated to ETP. Thus, the induction hypothesis holds for the next step.

Note that this proof also shows that an instance which satisfies SPC and ETP is consistent.

Finally, given the ordering $<$, we show that finding a solution can be done in polynomial time. Given a consistent assignment $(u_1, u_2, \ldots u_l)$ with $l < n$, finding a compatible value u_{l+1} for the next variable x_{l+1} is feasible by searching in its domain whose the size is at most d. For each value, we need to verify the constraints connecting the variable x_{l+1} which can be done in $O(e_{l+1})$ if the next variable x_{l+1} has e_{l+1} neighbors in the previous variables. Since $\Sigma_{1\leq l<n} e_{l+1} = e$, the total cost to find a solution is $O((n+e).d)$. □

As a consequence of theorems 2 and 3, the class of instances satisfying ETP and which are SPC is a tractable class. Moreover, we can extend Theorem 1:

Theorem 4 *If ETP-SPC (resp. BTP-SPC) denotes the class of instances satisfying ETP (resp. BTP) and SPC, then*

- $BTP\text{-}SPC \subsetneq ETP\text{-}SPC \subsetneq ETP$
- $BTP\text{-}SPC \subsetneq BTP \subsetneq ETP$.

One of the most interesting properties of the tractable class BTP is the fact that the instances of this class can be solved in polynomial time using classical algorithms (such as MAC or RFL) implemented in most solvers. The next properties establishes that a similar result holds for ETP. Indeed, the proof of Theorem 3 allows to show that an algorithm such as BT (Backtracking) can solve any instance of the class ETP-SPC in polynomial time:

Theorem 5 *If a binary CSP instance P satisfies SPC and ETP with respect to a variable ordering $<$, the algorithm BT finds a solution of the instance P in polynomial time.*

Proof: As the instance satisfies SPC, BT using the ordering $<$ for the variable assignment can find a consistent assignment on x_1, x_2 and x_3. Moreover, it is shown that a consistent assignment $(u_1, u_2, \ldots u_{l-1}, u_l)$ on $x_1, x_2, \ldots x_{l-1}$ and x_l can be extended to a $(l+1)^{th}$ variable, that is on x_{l+1}. To find the assignment of x_{l+1}, we need to look for a compatible value in its domain. This is feasible in $O(e_{l+1}.d)$ assuming that x_{l+1} has e_{l+1} neighbors in the previous variables. So, as for the proof of Theorem 3, finding a solution of P is globally feasible in $O((n+e).d)$. □

Note that by the same reasoning, we can show that a solution of a binary CSP instance satisfying SPC and ETP can be found in $O(n.(n+e).d^2)$, using the same variable ordering, by the algorithms MAC and RFL, where the additional cost is due to the arc-consistency filtering performed after each variable assignment.

Corollary 1 *If a binary CSP instance P satisfies SPC and ETP with respect to a variable ordering $<$, the algorithms MAC and RFL find a solution of the instance P in polynomial time.*

Nevertheless, a question remains open concerning the use of an ordering which does not satisfy ETP for the variable assignment for algorithms MAC or RFL while these algorithms execute in polynomial time (despite the use of a different order) for the instances of the class BTP.

In the next section, we discuss the interest of the class ETP from a practical viewpoint We will analyze in particular its presence in the instances, but also we will check whether the instances need to verify SPC to be efficiently solved in practice.

Experimentations

Now, we wonder whether some instances usually exploited as benchmarks for solver comparisons satisfy the ETP property. With this aim in view, we consider 2,260 binary benchmarks of the CSP 2008 Competition[3].

Among these 2,260 instances, we have observed that the ETP property occurs more frequently than the BTP one. Indeed, 41 instances satisfy the ETP property while only 13 satisfy the BTP property. However, only two instances satisfy both ETP and SPC because, generally, for the other instances, some of their values make them arc-inconsistent or path-inconsistent. In order to avoid this problem and in the same spirit as the hidden tractable classes introduced in (El Mouelhi, Jégou, and Terrioux 2014), a natural solution consists in achieving an AC or SPC filtering as pre-processing step. When enforcing AC, 236 instances are detected as inconsistent and so for them, BTP and ETP hold trivially. Among the other instances, for 50 instances, the obtained simplified instance satisfies the ETP property (against 46 for BTP). For 36 of them, it is also path-consistent and so belongs to the tractable class ETP-SPC. Table 1 provides the some instances which are BTP and/or ETP before or after the application of the AC filtering. Now, if we enforce SPC, 628 instances are detected as inconsistent and so are trivially BTP and ETP. For 76 instances, the obtained simplified instance satisfies ETP against 71 for BTP. These instances highlight the status of hybrid class. Indeed, 8 instances (e.g. hanoi-3_ext or graph12-w0) belong to ETP-SPC thanks to their particular structure (i.e. their constraint graph is acyclic) while the others like domino-100-100 belong to ETP-SPC due to their particular relations. Moreover, we can also note the diversity of these instances (academic, random or real-world instances).

Regarding the solving, we have established in the previous section that MAC and RFL are able to solve the instances of the class ETP-SPC in polynomial time if they exploit the variable ordering used in the ETP property. In practice, all the instances satisfying the ETP property (which belong or not to the class ETP-SPC) are solved in a backtrack free-manner by MAC and RFL with the dom/wdeg heuristic (Boussemart et al. 2004) or a random variable ordering.

More generally, we found 198 other binary instances which are not detected as inconsistent by AC and are solved in a backtrack free manner by MAC (we make a similar observation for RFL). In order to provide information about these instances, we consider the notion of backdoor (Williams, Gomes, and Selman 2003). A *backdoor* is a set of variables defined w.r.t. a particular algorithm such that once the backdoor variables are assigned, the problem becomes easy under that algorithm. This particular algorithm can be, for instance, any algorithm which solves the instances of a given tractable class in polynomial time. If we consider the binary instances which have an acyclic constraint graph, MAC is able to solve them in polynomial time without any additional processing. Then, MAC is also known to implicitly exploit cycle-cutsets (i.e. a particular case of backdoor) (Sabin and Freuder 1997). A cycle-cutset is a subset X_{CC} of variables such that the subproblem induced by X_{CC}[4] has an acyclic constraint graph. After having assigned some variables (i.e. the variables of a cycle-cutset of the constraint graph), the remaining part of the problem becomes acyclic and so MAC solves this part in polynomial time. Unlike the Cycle-Cutset method (Dechter 1990), for which the cycle-cutset is generally computed before running the algorithm, MAC is able to exploit such a set implicitly during the search. For an instance P of the ETP class, we can have the same reasoning with a subset X' of variables such that once removed thanks to variable assignments, the problem $P_{|X'}$ has the ETP property. Computing the smallest subset X' such that $P_{|X'}$ has the ETP property is NP-hard (the proof is similar to one for BTP (Cooper, Jeavons, and Salamon 2010)). But, here, this is not a problem, since we do not have to compute such a smallest subset. Indeed, by assigning variables, MAC will implicitly discover a subproblem $P_{X'}$ which is ETP. The subset X' corresponds then to the subset of the variables assigned by MAC. Of course, we cannot have any guarantee about the optimality of the subset X' discovered by MAC. In order to assess the size of these subsets, we run MAC by checking at each step whether the ETP property holds for the remaining part of the problem. We performed the same test for BTP and for both ETP and SPC. As these experimentations are very time expensive, we focus our study on 102 instances among the 198 instances which are consistent and not ETP.

We first observe that the size of the subset is generally smaller for ETP than for BTP. Such a result was foreseeable since ETP generalizes BTP. In most cases, the size of the subset for ETP and SPC is close to the one for ETP and smaller than the one for BTP. Table 2 gives the number of variables which must be assigned by MAC before the remaining part of the problem is BTP, ETP or both ETP and SPC for some instances. The selected instances are representative of the observed trends. For 32 instances (respectively 20), MAC requires to assign less than the third of the variables in order that the remaining part of the problem is ETP (resp. ETP and SPC) while the same result for BTP is only observed for 12 instances. Furthermore, for 12 instances (resp. 8), after at most the third assignment, the remaining part of the problem becomes ETP (resp. ETP

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

[4] The subproblem of an instance $P = (X, D, C)$ induced by a subset X' of X is the instance $P_{|X'} = (X - X', D', C')$ where $D' = \{D_{x_i} \in D | x_i \in X - X'\}$ and $C' = \{c \in C | S(c) \subseteq X - X'\}$.

Instance	BTP	ETP	BTP	ETP
	before AC		after AC	
crossword-m1-uk-puzzle01	no	no	yes	yes
domino-100-100	no	yes	yes	yes
fapp17-0300-4	no	yes	no	yes
graph12-w0	yes	yes	yes	yes
hanoi-3_ext	yes	yes	yes	yes
pigeons-50-ord	yes	yes	yes	yes

Table 1: Some instances which are BTP and/or ETP before or after the application of the AC filtering.

Instance	n	BTP	ETP	ETP-SPC
2-insertions-3-4	37	17	1	1
anna-11	138	36	29	29
bqwh-15-106-35_ext	106	90	33	90
mug100-25-4	100	45	3	3
myciel3-4	11	4	1	1
super-os-taillard-4-11	32	24	20	21

Table 2: Number of variables which must be assigned by MAC before the remaining part of the problem is BTP, ETP or both ETP and SPC for some instances.

and SPC). So, in such a case, we can consider that the presence of the ETP property inside the instances may mainly explain the good efficiency of the MAC algorithm on these instances. In contrast, if many variables must be assigned in order to make the remaining part of the problem ETP, we have to study in a deeper manner the behavior of MAC in order to provide explanations of its practical efficiency.

Conclusion

We have defined a new property called ETP, which constitutes a generalization of the Broken Triangle Property (BTP). Thanks to ETP, we have introduced a new tractable class of CSPs called $ETP\text{-}SPC$, defined by instances which satisfy both ETP and Strong-Path-Consistency. This work seems useful because BTP is one of the most important tractable class in the state of the art of CSPs. This new class has the same type of desirable properties as BTP, in particular it can be solved in polynomial time, directly by solvers using standard algorithms such as MAC or RFL. In addition to the theoretical contribution of this work, we have experimentally shown the presence of this class among the benchmark instances of the CSP community. We have also shown the potential interest of its use for the solving of any instances thanks to the notion of backdoors. A first possible extension of this work would be to relax the required satisfaction of path-consistency to ensure the tractability of the instances satisfying the ETP property. For example, directional path consistency is an immediate relaxation but we think to less powerful local consistencies than ones based on path consistency. Moreover, we naturally think of extending this tractable class to CSPs with constraints of any arity. It would also be useful to analyze the relationships of this class with other tractable classes of CSPs.

Acknowledgments

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

Boussemart, F.; Hemery, F.; Lecoutre, C.; and Sais, L. 2004. Boosting systematic search by weighting constraints. In *ECAI*, 146–150.

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

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

Dechter, R. 1990. Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence* 41:273–312.

El Mouelhi, A.; Jégou, P.; and Terrioux, C. 2014. Hidden Tractable Classes: from Theory to Practice. In *ICTAI*, 437–445.

Freuder, E. 1982. A Sufficient Condition for Backtrack-Free Search. *JACM* 29 (1):24–32.

Gottlob, G.; Leone, N.; and Scarcello, F. 2000. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence* 124:343–282.

Haralick, R., and Elliot, G. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14:263–313.

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

Jégou, P. 1993. Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In *AAAI*, 731–736.

Montanari, U. 1974. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Artificial Intelligence* 7:95–132.

Nadel, B. 1988. *Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms*. In *Search in Artificial Intelligence*. Springer-Verlag. 287–342.

Rossi, F.; van Beek, P.; and Walsh, T. 2006. *Handbook of Constraint Programming*. Elsevier.

Sabin, D., and Freuder, E. 1994. Contradicting Conventional Wisdom in Constraint Satisfaction. In *ECAI*, 125–129.

Sabin, D., and Freuder, E. 1997. Understanding and Improving the MAC Algorithm. In *CP*, 167–181.

Williams, R.; Gomes, C. P.; and Selman, B. 2003. Backdoors to typical case complexity. In *IJCAI*, 1173–1178.