Hidden Tractable Classes: from Theory to Practice

Achref El Mouelhi Philippe Jégou Cyril Terrioux
Aix-Marseille Université, LSIS UMR 7296
13397 Marseille Cedex 20 (France)
{*achref.elmouelhi, philippe.jegou, cyril.terrioux*}*@lsis.org*

***Abstract*—Tractable classes constitute an important issue in CP, at least from a theoretical viewpoint. But they are not actually used in practice. Either their treatment is too costly for time complexity or, even if there exist efficient algorithms to manage them, they do not appear in the real problems. We propose here to address this issue thanks to the notion of *hidden tractable classes*. Such classes are based on a known tractable class $\mathcal{C}$, and a transformation t, and are defined by sets of instances P such that their transformation using t is in $\mathcal{C}$, that is $t(P) \in \mathcal{C}$. We propose a general framework to study such notions. After, we focus our study on the tractable class BTP, and several transformations which are the filterings classically used in CP. We show then that the use of filterings allows sometimes to highlight the occurrence of BTP in the benchmarks generally considered for solver comparisons, i.e. that BTP is sometimes "hidden" in the benchmarks. Thus, this approach allows to extend the set of known tractable classes.**

I. Introduction

Constraint Satisfaction Problems (CSPs [1]) 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 [2], RFL (for Real Full Look-ahead [3]) or MAC (for Maintaining Arc Consistency [4]) 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 a theoretical interest only, without allowing to improve solving capabilities in practice. So, these classes are not in fact exploited 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 or particular processing). For example, this is the case of the BTP class [5] 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.

Here, we choose to study the presence of tractable classes in benchmarks. The difficulty is to highlight them. A way which, to our knowledge, has not been studied yet is to highlight the presence of these classes using the same filtering techniques as ones exploited in the solvers at each step or as pre-processing. In addition, we believe that this allows, after filtering, to highlight the belonging to tractable classes of the simplified instances. Such classes will be called *hidden tractable classes*. To study them, we introduce a formal framework based on the notion of transformation of instances. Given a class $\mathcal{C}$ and a transformation t, we define the notion of hidden class $\mathcal{C}$ discoverable by a transformation t. The instances of such classes are instances P such that $t(P) \in \mathcal{C}$. So, if the cost of computing t is polynomial, t allows to extend the class $\mathcal{C}$. The formal framework we define allows us to analyse the space of such classes w.r.t. different classes and different transformations.

To show the interest of such an approach, we will focus our study on the analysis of the class BTP. This class possesses several interesting properties in this context. Firstly, it includes several tractable classes previously defined in the literature. So, it is a class that can occur more easily in the benchmarks than some other classes more restrictive. In

addition, it has algorithmic properties that should facilitate its implicit handling by solvers. On the one hand, a standard CSP solver based on algorithms such as MAC or RFL can solve any instance of BTP directly and efficiently without using specific algorithms. On the other hand, it is well suited to filtering techniques as the ones achieving reductions of domains since it is conservative, which means that an instance belonging to BTP which is filtered turns into a new simplified instance that still belongs to BTP. From a practical viewpoint, we show that some benchmarks usually exploited for solver comparisons belong to BTP or to one of its hidden class.

The paper is organized as follows. Section II introduces notations and recalls some notions about filtering and tractable classes. Section III presents our formal framework. Then we illustrate it by considering the BTP class in section IV before concluding.

II. Background

A. Notations

Formally, a *constraint satisfaction problem* 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 $|S(c_i)|$. A CSP is called *binary* if all constraints are of arity 2. In this paper, for sake of simplicity, we only deal with the case of binary CSPs but the framework we propose can be easily generalized to constraints of any arity in order to handle tractable classes involving non-binary CSPs. Hence, we will denote by c_{ij} the constraint 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 jointly with heuristics, constraint learning, non-chronological backtracking, filtering techniques based on constraint propagation, etc. The worst time complexity for these algorithms is naturally exponential, at least in $O(e.d^n)$ where n is the number of variables and d the maximum size of domains. Despite this exponential complexity, the solvers have shown in many cases their practical efficiency. This requires in any case the implementation of filtering techniques which are based on the concept of local consistency.

B. Local Consistencies and Filtering

The most popular and oldest local consistency is called *arc-consistency* (AC). Given a binary CSP $P = (X, D, C)$, a value $v_i \in D(x_i)$ is arc-consistent w.r.t. $c_{ij} \in C$ iff there exists a value $v_j \in D(x_j)$ s.t. $(v_i, v_j) \in R(c_{ij})$. Then, $v_j \in D(x_j)$ is a *support* of v_i for the constraint c_{ij}. A domain $D(x_i)$ is arc-consistent w.r.t. c_{ij} iff $\forall v_i \in D(x_i)$, the value v_i is arc-consistent w.r.t. c_{ij}, and the CSP P is arc-consistent iff $\forall D(x_i) \in D$, the domain $D(x_i)$ is arc-consistent w.r.t. all $c_{ij} \in C$. A filtering of domains based on AC consists in removing the values which do not satisfy the arc-consistency. AC-2001 [6] is one of the most efficient algorithm for enforcing arc-consistency. Its time complexity is O($e.d^2$). It is really efficient in practice and then it can also be used during search. Nevertheless, the filtering power of AC can be really limited because of the local definition of the consistency. So, more powerful consistencies performing more powerful filterings have been defined. For example, Montanari has first defined a generalization which is called *path-consistency* (denoted *PC* [7]). Given two variables x_i and x_j, a pair of values $(v_i, v_j) \in D(x_i) \times D(x_j)$ is *path-consistent* iff for any third variable $x_k \in X$ with $c_{ik} \in C$ and $c_{jk} \in C$, there exists a value $v_k \in D(x_k)$ such that $(v_i, v_k) \in R(c_{ik})$ and $(v_j, v_k) \in R(c_{jk})$. The value v_k is then called a *support* of the tuple (v_i, v_j). Finally, the CSP is path consistent if any pair of variables (x_i, x_j) with $i \neq j$ is path consistent. Note that the filtering associated to PC deletes tuples in relations (but no value in domains) and thus, can add new binary constraints in the CSP since if a binary constraint c_{ij} does not appear in an instance, we consider that the associated relation is the universal relation. PC can be efficiently achieved using, for example, the algorithm PC-2001 [6], its time complexity being O($n^3.d^3$). Since, this filtering does not reduce domains, we can enforce first path-consistency, and after arc-consistency, using the local consistency called *strong-path-consistency (SPC)* in O($n^3.d^3$). Nevertheless, the practical cost of the associated filtering is sometimes unrealistic. So, other local consistencies has been defined, as for example, the *max-restricted path-consistency* (denoted *maxRPC* [8]). A value $v_i \in D(x_i)$ is $maxRPC$ iff for every constraint $c_{ij} \in C$ there exists a tuple $(v_i, v_j) \in R(c_{ij})$ such that for every additional variable x_k, there exists a value $v_k \in D(x_k)$ such that if $c_{ik} \in C$, $(v_i, v_k) \in R(c_{ik})$ and if $c_{jk} \in C$, $(v_j, v_k) \in R(c_{jk})$. The cost of achieving $maxRPC$ is O($e.n.d^2$) [8]. Another way to define local consistencies is based on the subproblem induced by an assignment $x_i = v_i$. For example, a CSP is *singleton arc-consistent* (denoted *SAC* [9]) if for all domains and then all their values $v_i \in D(x_i)$, the subproblem induced by the assignment $x_i = v_i$ has (non-empty) arc-consistent sub-domains. The time complexity is O($e.n.d^3$) [10]. To limit the cost of filtering, the notion of *inverse consistency* has been proposed in [11]. For example, for

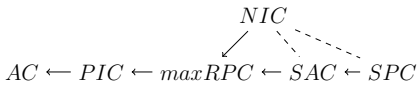

Figure 1. Relationship between consistencies.

the *neighborhood-inverse consistency* (denoted *NIC* [11]), the filtering of a domain is induced by the compatibility of values of the associated variable w.r.t. the subproblem defined by its neighborhood in the network. So, the time complexity is related to the maximum degree Δ of a variable in the constraint network, namely $O(\Delta^2.(n + e.d).d^{\Delta+1})$. Unfortunately, if Δ is large, the time cost can be prohibitive. So, a limited version of *NIC* as been proposed with *path-inverse consistency* (denoted *PIC* [11]). A value $v_i \in D(x_i)$ is PIC iff for every pair of additional variables x_j and x_k, there exist values $v_j \in D(x_j)$ and $v_k \in D(x_k)$ such that (v_i, v_j, v_k) is a consistent partial assignment. The cost of achieving *PIC* is $O(en + ed^2 + cd^3)$ with c the number of 3-cliques in the constraint graph using the algorithm PIC2 [12]. Note that AC, SAC, NIC, PIC and $maxRPC$ realize domain filterings while SPC filters both domains and constraints since it can delete tuples from relations, as PC does.

An analysis given in [9] presents the comparison between numerous local consistencies. The comparison is based on formal relations between consistencies; we recall them. We say that a consistency CO_1 is *stronger* than a consistency CO_2 (denoted $CO_2 \leq CO_1$) if in any CSP instance P in which CO_1 holds, CO_2 holds too. So, any algorithm achieving CO_1 deletes at least the values removed by CO_2. We say that a consistency CO_1 is *strictly stronger* than a consistency CO_2 (denoted $CO_2 < CO_1$) if $CO_2 \leq CO_1$ and there is at least one CSP instance P in which CO_2 holds and CO_1 does not. Note that these relations are transitive. Finally, we say that CO_1 and CO_2 are *incomparable* if neither relation between them hold. Figure 1 summarizes some relations between consistencies: an arc from CO_2 to CO_1 (resp. a dashed line between CO_1 and CO_2) means that $CO_1 < CO_2$ (resp. CO_1 and CO_2 are incomparable).

C. Tractable Classes

Although the problem CSP is NP-complete, there exist classes of instances that can be solved in polynomial time. These classes are called "tractable classes" and rely on some properties that can be verified by the instances.

Definition 1 (Tractable class): A *tractable class* for CSP is a set (possibly infinite) $\mathcal{C}$ of instances of CSP such that there exists two polynomial time algorithms A_R and A_S such that for any instance P, A_R can decide if $P \in \mathcal{C}$ and, if $P \in \mathcal{C}$, A_S solves P.

Here, we consider the definition proposed in [13] but in the literature, some authors consider only the fact that a solving algorithm A_S exists.

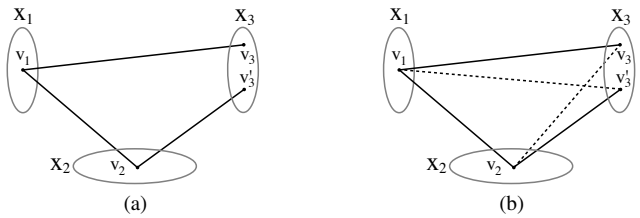

Figure 2. A non-BTP pattern (a) and a BTP one (b) w.r.t. the order $x_1 < x_2 < x_3$.

For CSPs, 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 binary CSPs can be solved 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" [15]. More recently, new tractable classes have been defined which are related to these two kinds of properties, such as the BTP class [5]. 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 2 (Broken Triangle Property [5]): 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, (v_i, v_j), (v_i, v_k) and (v_j, v'_k) 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 can be graphically visualized on the microstructure graph[1]. E.g. in Figure 2 (a), there is a broken triangle on x_3 with respect to 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 in Figure 2 (b), if one of the two dashed edges (that is binary tuples) appears in the microstructure, the BTP property holds independently from the ordering.

III. Hidden Tractable Classes

The study of tractable classes is an important issue in CP, at least from a theoretical point of view. But the tractable classes are not actually explicitly used in practice, or rarely. Sometimes, their treatment is too costly for time. Or, even if they can be easily exploited (e.g. in linear time), they seem to not really appear in real problems, and thus, it is not easily possible to exploit them with solvers. We propose here to address this issue thanks to the use of the filterings generally

[1]The micro-structure [16] of a binary CSP $P = (X, D, C)$ is the 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 $(v_i, v_j) \in R(c_{ij})\}$.

used in CP solvers as AC or more powerful filterings. Indeed, some tractable classes can be sometimes "hidden" in the instances while they could be "discovered" by the means of filterings. To this end, we propose in this section, a general framework which can be used for any tractable class and any kind of transformation of instances, as the filterings for example.

Definition 3: Given an instance $P = (X, D, C)$, t is called a *transformation* of P if $t(P) = (t_{var}(X), t_{dom}(D), t_{cons}(C))$ verifies:

- $t_{var}(X) \subseteq X$
- $t_{dom}(D) = (t_{dom}(D(x)) : x \in t_{var}(X)$ *and* $t_{dom}(D(x)) \subseteq D(x))$
- $\forall c_i \in C, t_{cons}(c_i)$ verifies:
  - $t_{cons}(S(c_i)) = S(c_i) \backslash \{x \in X : x \notin t_{var}(X)\}$ and
  - $t_{cons}(R(c_i)) \subseteq R(c_i)[t_{cons}(S(c_i))]$ with $R(c)[Y]$ the projection of $R(c)$ to the variables of Y.
- $\forall c' \in t_{cons}(C)$ such that $c' \notin C$, $t_{cons}(R(c')) \subseteq \prod_{x' \in S(c')} t_{dom}(D(x'))$.

Note that if a constraint c' of $t_{cons}(C)$ is not in C, its relation is a subrelation of the universal relation associated to the scope $S(c')$ which is implicitly defined in P.

As classical transformations of instances, we find generally the deletion of variables, the deletion of values, the addition of constraints, and the deletion of tuples in compatibility relations of constraints. Filterings correspond to the deletion of values but also to the addition of constraints with deletion of tuples. Moreover, the assignments of variables can be considered as particular cases of filterings since an assignment $x_i = v_i$ can be considered as the filtering achieving $t_{dom}(D(x_i)) = \{v_i\}$. Here, we have considered transformations defined by simplifications of instances. But we could define transformations more generally, e.g. based on the addition of new variables and new values in the domains, as well as the addition of tuples in existing relations or the removal of constraints. Finally, note that some classical properties of the transformations of instances, as the preservation of the satisfiability or the preservation of the set of solutions are not required here.

Definition 4: Given a transformation t and a set (also called a class) of instances $\mathcal{P}$ we have $t(\mathcal{P}) = \{t(P) : P \in \mathcal{P}\}$.

So, given a transformation t and a class of instances $\mathcal{C}$, it is possible that an instance P which does not belong to $\mathcal{C}$ appears in $\mathcal{C}$ after a transformation obtained by t. The belonging of a such instance will be then *brought to light* by t:

Definition 5: Given a transformation t and a class of instances $\mathcal{C}$, the class of instances *brought to light by* t for $\mathcal{C}$ is $\mathcal{C}^t = \{P : t(P) \in \mathcal{C}\}$.

So, we can introduce the notion of *hidden (tractable) class*:

Definition 6: Given a set $\mathcal{P}$ of instances of CSP, a transformation t and a class $\mathcal{C}$, $\mathcal{P}$ is called *hidden class* of $\mathcal{C}$ for t, if $t(\mathcal{P}) \subseteq \mathcal{C}$, that is if $\mathcal{P} \subseteq \mathcal{C}^t$. $\mathcal{P}$ is called *hidden tractable class* of $\mathcal{C}$ for t if t preserves the consistency and can be achieved in polynomial time and C is tractable.

In case an instance $P \in \mathcal{P}$ such that $\mathcal{P}$ is a hidden class of $\mathcal{C}$ for a filtering t, and if $\mathcal{C}$ is a tractable class, it is then sufficient to apply the filtering t to achieve a simplified version of P which then belongs to a tractable class. Assuming that the cost of the filtering t is polynomial, $\mathcal{P}$ is thus a tractable class. This kind of approach is particularly adapted to the solvers of the state of the art because they generally use filterings during search.

Like for local consistencies and associated filterings, we can define comparisons between classes modified by the mean of transformations.

Definition 7: Given two transformations t_1 and t_2, and two classes $\mathcal{C}_1$ and $\mathcal{C}_2$, we say that $\mathcal{C}_2^{t_2}$ is *larger* than $\mathcal{C}_1^{t_1}$ (denoted $\mathcal{C}_1^{t_1} \leq \mathcal{C}_2^{t_2}$) if $\mathcal{C}_1^{t_1} \subseteq \mathcal{C}_2^{t_2}$. Moreover, we say that $\mathcal{C}_2^{t_2}$ is *strictly larger* than $\mathcal{C}_1^{t_1}$ (denoted $\mathcal{C}_1^{t_1} < \mathcal{C}_2^{t_2}$) if $\mathcal{C}_1^{t_1} \subsetneq \mathcal{C}_2^{t_2}$, and we say that $\mathcal{C}_1^{t_1}$ and $\mathcal{C}_2^{t_2}$ are *incomparable* (denoted $\mathcal{C}_1^{t_1} \perp \mathcal{C}_2^{t_2}$) if neither relation between them holds. Finally, we say that $\mathcal{C}_1^{t_1}$ and $\mathcal{C}_2^{t_2}$ are *equal* (denoted $\mathcal{C}_1^{t_1} = \mathcal{C}_2^{t_2}$) if $\mathcal{C}_1^{t_1} \leq \mathcal{C}_2^{t_2}$ and $\mathcal{C}_2^{t_2} \leq \mathcal{C}_1^{t_1}$.

Property 1: For any classes $\mathcal{C}_1$ and $\mathcal{C}_2$ such that $\mathcal{C}_1 \subseteq \mathcal{C}_2$, and for any transformation t, we have $\mathcal{C}_1^t \leq \mathcal{C}_2^t$.

Proof: Let $P \in \mathcal{C}_1^t$. By definition, $t(P) \in \mathcal{C}_1$. Since $\mathcal{C}_1 \subseteq \mathcal{C}_2$, we have $t(P) \in \mathcal{C}_2$, and then $P \in \mathcal{C}_2^t$. Thus $\mathcal{C}_1^t \subseteq \mathcal{C}_2^t$, and we have $\mathcal{C}_1^t \leq \mathcal{C}_2^t$. □

Note that if $\mathcal{C}_1 \subsetneq \mathcal{C}_2$, we have not necessarily $\mathcal{C}_1^t < \mathcal{C}_2^t$. For example, if t is the transformation of binary CSPs that computes the *minimal CSP* [7] of any binary instance, and if $\mathcal{C}_1$ is the class of minimal binary CSPs while $\mathcal{C}_2$ is the class of arc-consistent binary CSPs (including CSPs with empty domains), we have $\mathcal{C}_1 \subsetneq \mathcal{C}_2$ because any minimal CSP is arc-consistent, while $\mathcal{C}_1^t = \mathcal{C}_2^t$.

The aim of the study of transformations is to highlight classes of instances that become tractable once transformed. To be suitable for implementation in solvers that perform a sequence of transformations (e.g. AC filtering in MAC or RFL), it would be desirable to use transformations that do not endanger the properties of the considered tractable class. In other words, we want to exclude the transformations t such that given a tractable class $\mathcal{C}$ and an instance $P \in \mathcal{C}$, we have $t(P) \notin \mathcal{C}$. This question has been addressed in [5] using the definition of classes which are *conservative* with respect to the restriction of domains (precisely closed under domain restriction). We generalize this notion to any transformation.

Definition 8: A class $\mathcal{C}$ of instances of CSP is called *conservative* w.r.t. a transformation t if it is closed for t, i.e. $\forall P \in \mathcal{C}, t(P) \in \mathcal{C}$ (or $t(\mathcal{C}) \subseteq \mathcal{C}$).

Some properties can be directly inferred from this notion:

Property 2: Any class of instances $\mathcal{C}$ is conservative w.r.t. the identity mapping (denoted Id), that is $Id(\mathcal{C}) \subseteq \mathcal{C}$.

In [5], this concept has been used in the case of domain filtering:

Property 3 ([5]): The tractable class BTP is conservative w.r.t. the filtering of domains.

Beyond the filterings, this concept can also be exploited by considering the deletions of variables related to structural properties of constraint networks:

Property 4: The class $TREE$ (defined as the set of acyclic binary CSPs) is conservative w.r.t. any transformation of instances which deletes some variables (vertices in the constraint graph).

Other properties can be inferred using the comparison between classes modified by the mean of transformations.

Property 5: If $\mathcal{C}$ is conservative w.r.t. a transformation t, then $\mathcal{C} \subseteq \mathcal{C}^t$.

Proof: Let $P \in \mathcal{C}$. Since $\mathcal{C}$ is conservative for t, $t(P) \in \mathcal{C}$, and thus $P \in \mathcal{C}^t$. □

Corollary 1: If $\mathcal{C}$ is conservative w.r.t. a transformation t, then $\mathcal{C}^{Id} \leq \mathcal{C}^t$.

Proof: Since $\mathcal{C} \subseteq \mathcal{C}^t$ and $\mathcal{C}^{Id} = \mathcal{C}$, we have $\mathcal{C}^{Id} \subseteq \mathcal{C}^t$. □

If we consider some kinds of transformations, one can deduce more specific properties, as is the case for filtering:

Property 6: Let t_1 and t_2 be two transformations which are filterings such that $t_1 \leq t_2$. For any class $\mathcal{C}$ which is conservative w.r.t. t_1 and t_2, we have $\mathcal{C}^{t_1} \leq \mathcal{C}^{t_2}$.

Proof: Let $P \in \mathcal{C}^{t_1}$, we show that $P \in \mathcal{C}^{t_2}$.
Since $P \in \mathcal{C}^{t_1}$, we have $t_1(P) \in \mathcal{C}$. Moreover, since $t_1 \leq t_2$, $t_2(t_1(P)) = t_2(P)$. So, since $t_1(P) \in \mathcal{C}$, and since $\mathcal{C}$ is conservative w.r.t. t_2, $t_2(t_1(P)) \in \mathcal{C}$, and thus $t_2(P) = t_2(t_1(P)) \in \mathcal{C}$. Thus $P \in \mathcal{C}^{t_2}$. □

One can note that this property is not verified for strict relations. Indeed, we can define classes of instances $\mathcal{C}$ which are conservative w.r.t. t_1 and t_2 such that $t_1 < t_2$, while $\mathcal{C}^{t_1} < \mathcal{C}^{t_2}$ is false. E.g. if $\mathcal{C}$ is the class of all CSPs, and if $t_1 = AC$ and $t_2 = SAC$, by property 5, we have $\mathcal{C}^{t_1} = \mathcal{C}$ and $\mathcal{C}^{t_2} = \mathcal{C}$.

Although some general properties cannot hold, we can give some examples of such relations with different transformations and different classes:

Example 1 ($\mathcal{C}^{t_1} < \mathcal{C}^{t_2}$): Consider $\mathcal{C} = BTP$ which is conservative w.r.t. $t_1 = Id$ and $t_2 = AC$. We know that $t_1 < t_2$, but we have $\mathcal{C}^{t_1} < \mathcal{C}^{t_2}$, that is $BTP^{Id} < BTP^{AC}$.

Note that Id is not a local consistency, but we can define for the associated transformation the *empty property* which is universally satisfied.

Example 2 ($\mathcal{C}_1^t < \mathcal{C}_2^t$): Consider $\mathcal{C}_1 = TREE$ and $\mathcal{C}_2 = BTP$. From [5], we know that the class $TREE \subsetneq BTP$. Moreover, $TREE$ and BTP are conservative w.r.t. $t = AC$. We have $\mathcal{C}_1^t < \mathcal{C}_2^t$, that is $TREE^{AC} < BTP^{AC}$.

Example 3 ($\mathcal{C}_1^{t_1} = \mathcal{C}_2^{t_2}$): Here $t_1 = DCC$ is any transformation of instances which deletes the vertices of a cycle-cutset of binary CSPs and $t_2 = Id$, and $\mathcal{C}_1 = TREE$ while $\mathcal{C}_2 = CSP$, the set of all possible instances of binary CSPs. We have $\mathcal{C}_1^{t_1} \leq \mathcal{C}_2^{t_2}$, that is $TREE^{DCC} \leq CSP^{Id}$. But we have also $\mathcal{C}_2^{t_2} \leq \mathcal{C}_1^{t_1}$ that is $CSP^{Id} \leq TREE^{DCC}$. Of course, if CSP^{Id} is a hidden class, it is not a hidden tractable class. Likewise, $TREE^{DCC}$ is a hidden class but not a hidden tractable class since DCC does not preserve the consistency.

The use of transformations such as DCC is close to the notion of backdoor [17]. 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. For example, once that the variables of a cycle-cutset are assigned, using a MAC-(or RFL-)like algorithm, the induced subproblem can be solved in linear time [18]. This approach is a first way to exploit tractable classes which are hidden or not. In this paper, we will consider another way to exploit hidden classes which is related to transformations based on filterings. While it could be possible to consider different classes, our study is based here on the class BTP because it is well known to be an important tractable class for the CSP problem.

IV. The case of BTP

A. Theoretical results

We first remind known results about the relationship of the classes RRM[2], $TREE$ and $DUALTREE$[3] introduced in [5] and the class BTP.

Theorem 1 ([5]): (i) $RRM^{Id} < BTP^{Id}$.
(ii) $TREE^{Id} < BTP^{Id}$.
(iii) $DUALTREE^{Id} < BTP^{Id}$.

Then, we are interested to the relationship between BTP and some of its hidden classes. Here, we focus our study on transformations based on filtering.

Theorem 2: (i) $BTP^{Id} < BTP^{AC} < BTP^{PIC} < BTP^{maxRPC} < BTP^{SAC}$.
(ii) $BTP^{maxRPC} < BTP^{NIC}$.
(iii) $BTP^{SAC} \perp BTP^{NIC}$.

Proof: Property 3 holds notably for AC. So we have $BTP^{Id} \leq BTP^{AC}$ according to Corollary 1. If we consider the instance $L_{2,2}$ of Langford's number problem[4], we can note that it is not BTP but is arc-inconsistent and so trivially belongs to BTP^{AC}. So $BTP^{Id} < BTP^{AC}$.

According to Properties 3 and 6 and the relationship between consistencies depicted in Figure 1, we have $BTP^{AC} \leq BTP^{PIC} \leq BTP^{maxRPC} \leq BTP^{SAC}$ and

[2]A binary CSP $P = (X, D, C)$ belongs to the *renamable right monotone* class (denoted RRM) w.r.t. a variable ordering $<$ if, for $2 \leq j \leq n$, each domain $D(x_j)$ can be ordered by $\preccurlyeq_j$ s.t. for each constraint c_{ij} of C with $x_i < x_j$, $\forall v_i \in D(x_i), v_j, v'_j \in D(x_j)$, if $(v_i, v_j) \in R(c_{ij})$ and $v_j \preccurlyeq_j v'_j$ then $(v_i, v'_j) \in R(c_{ij})$.

[3]A binary CSP belongs to the $DUALTREE$ class if it is the dual of a tree-structured instance.

[4]The instance $L_{k,m}$ of the Langford's number problem (for more details, see problem 024 at CSPLib [19]) consists in arranging k sets of numbers $\{1, \ldots, m\}$ such that each appearance of the number i is i numbers after the previous one.

$BTP^{maxRPC} \leq BTP^{NIC}$. If we consider the queens problem for 4 queens, it belongs to BTP^{PIC} but not to BTP^{AC}. So $BTP^{AC} < BTP^{PIC}$. Likewise we can consider the instance $L_{3,4}$ (respectively $L_{3,5}$ and $L_{2,14}$) to show $BTP^{PIC} < BTP^{maxRPC}$ (resp. $BTP^{maxRPC} < BTP^{SAC}$ and $BTP^{maxRPC} < BTP^{NIC}$).

$L_{2,5}$ is in BTP^{NIC} but not in BTP^{SAC}. Conversely, Figure 3(e) of [9] depicts an instance belonging to BTP^{SAC} but not to BTP^{NIC}. So $BTP^{SAC} \perp BTP^{NIC}$. □

We can note, as mentioned in the previous proof, that the relationship between BTP and its hidden classes are closely related to one between the consistencies depicted in Figure 1. As recalled in this figure, we also know that $SAC < SPC$. Unfortunately, this result cannot be exploited because, to do so, we need the class BTP to be conservative w.r.t. SPC, what is not the case. Indeed, achieving PC can delete tuples in relations, namely the pairs of values $(v_i, v_j) \in R(c_{ij})$ which have no support w.r.t. a variable x_k. So, this filtering can invalidate the BTP property. The next theorem formalizes this assertion.

Theorem 3: BTP is not conservative w.r.t. PC, neither to SPC.

Proof: This is proved using a counterexample with 4 variables, x_1, x_2, x_3 and x_4 s.t. the variable x_3 must be in last rank in the ordering to satisfy BTP (at least after x_1 and x_2). Figure 3 shows independent parts of its microstructure. First, in Figure 3 (a), we have two broken triangles on x_1 and x_2. For this we need 8 values (8 vertices in the microstructure) and 6 binary tuples (6 edges). Secondly, in Figure 3 (b), we have 4 values, and 5 binary tuples (5 edges). The dashed edges correspond to the edges which are necessary to satisfy BTP. Considering the ordering $x_4 < x_1 < x_2 < x_3$, it is easy to see that this (partial) instance satisfies BTP. However, a PC filtering will remove all tuples since no tuple has a support in the domain of x_4, and therefore, BTP will be satisfied after the filtering. So, we add values and tuples such that all edges of the microstructure are not deleted after a PC filtering, except the dashed edges. In this way, the two broken triangles in Figure 3 (a) will not be deleted and a new broken triangle on x_3 will appear in Figure 3 (b). Thus, there will be no ordering on x_1, x_2 and x_3 allowing to satisfy BTP. So, for each edge (except the dashed ones), we add two values, one in each domain of the two other variables. E.g. for an edge from x_1 to x_2, we add one value in the domain of x_3 and one value in the domain of x_4. Finally, we add 5 edges to connect these 4 values, achieving a 4-clique (with 6 edges) since they mutually satisfy PC. By this way, after a PC filtering, none of these 6 edges will be deleted. So, for the 9 edges which are not dashed, we add 9×2 vertices and 9×5 edges. Globally, we have a microstructure with $12 + (9 \times 2) = 30$ vertices and $11 + (9 \times 5) = 56$ edges.

This instance satisfies BTP before filtering. Indeed, it is sufficient to consider the ordering $x_4 < x_1 < x_2 < x_3$ where no broken triangle appears. By cons, a PC filtering will only remove the two dashed edges so that BTP is finally not satisfied, since their deletion creates a broken triangle on x_3 with respect to the variables x_1 and x_2 while the two broken triangles on x_1 and x_2 are kept. Hence BTP is not conservative w.r.t. PC.

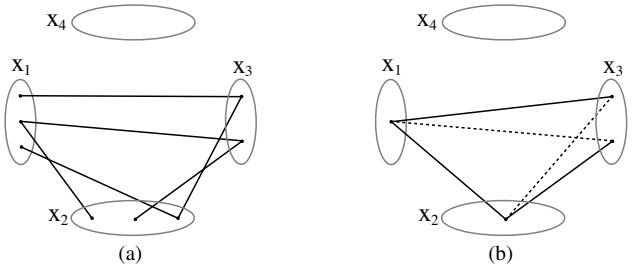

Figure 3. (a) Two broken triangles on x_1 and x_2, (b) a BTP pattern in x_3 thanks to the two dashed edges.

As the built instance is also arc-consistent, it is strong path-consistent. So BTP is not conservative w.r.t. SPC. □

So, as BTP is not conservative w.r.t. PC, neither SPC, we obtain the following theorem:

Theorem 4: (i) $BTP^{Id} \perp BTP^{(S)PC}$.
(ii) For any filtering $\phi \in \{AC, PIC, maxRPC, SAC, NIC\}$, $BTP^{\Phi} \perp BTP^{(S)PC}$.
(iii) $BTP^{PC} < BTP^{SPC}$.

Proof: The instance built in the previous proof allows to establish that $BTP^{Id} \not\subseteq BTP^{(S)PC}$. Likewise, for any filtering $\phi \in \{AC, PIC, maxRPC, SAC, NIC\}$, we have $BTP^{\Phi} \not\subseteq BTP^{(S)PC}$.

Conversely, the instance $L_{2,3}$ belongs to BTP^{PC} and BTP^{SPC} thanks to some tuple deletions. So it cannot belong to BTP^{Id} or to BTP^{Φ} for any filtering $\phi \in \{AC, PIC, maxRPC, SAC, NIC\}$. Hence we have $BTP^{Id} \perp BTP^{(S)PC}$ and $BTP^{\Phi} \perp BTP^{(S)PC}$.

Now we prove that $BTP^{PC} < BTP^{SPC}$. Let us consider an instance P of BTP^{PC}. By definition, $PC(P) \in BTP$. As it is well known that $SPC(P) = AC(PC(P))$ [20] and as BTP is conservative w.r.t. AC, $SPC(P) \in BTP$. Hence $BTP^{PC} < BTP^{SPC}$. □

Now we establish the relationship between BTP and $DBTP$[5] which has been proposed in [21] to extend the BTP property to non-binary CSPs.

Theorem 5: (i) $BTP^{Id} \perp DBTP^{Id}$.
(ii) $BTP^{AC} \perp DBTP^{AC}$.
(iii) $DBTP^{Id} < DBTP^{AC}$.

Proof: Theorem 11 of [21] states precisely that $BTP^{Id} \perp DBTP^{Id}$.

A consequence of Lemma 5 of [21] is that any arc-consistent binary instance which satisfies BTP and has two broken triangles for two different variables of a same triple

[5]A CSP 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. $\prec$.

of variables cannot satisfy DBTP [21]. Hence $BTP^{AC} \not\subseteq DBTP^{AC}$. Conversely, any arc-consistent DBTP instance with at least one non-binary constraint cannot belong to BTP^{AC}. So $BTP^{AC} \perp DBTP^{AC}$.

As $DBTP$ is conservative w.r.t. AC (Property 1 of [21]), we have $DBTP^{Id} \leq DBTP^{AC}$ according to Corollary 1. As $L_{4,3}$ belongs to $DBTP^{AC}$, but not to $DBTP^{Id}$, we have $DBTP^{Id} < DBTP^{AC}$. □

These results are summed up by Figure 4 where an arc from $\mathcal{C}_2^{t_2}$ to $\mathcal{C}_1^{t_1}$ (resp. a dashed line between $\mathcal{C}_1^{t_1}$ and $\mathcal{C}_2^{t_2}$) means that $\mathcal{C}_1^{t_1} < \mathcal{C}_2^{t_2}$ (resp. $\mathcal{C}_1^{t_1} \perp \mathcal{C}_2^{t_2}$).

Finally, we consider the solving of instances from BTP by classical algorithms like MAC or RFL. Theorem 7.6 of [5] states that MAC solves any instance of BTP in polynomial time. This result also holds for RFL since it only needs AC to be enforced at each step of the search. It can be also extended to the BTP^{AC} hidden class.

Theorem 6: MAC (resp. RFL) solves any instance from BTP^{AC} in polynomial time without having to enforce AC as pre-processing.

Proof: Let us consider an instance P from BTP^{AC}. We can remark that assigning a value v to a variable x can be seen as removing all the values except v from the domain of x and that, after removing any values from P and enforcing AC, the resulting instance is necessarily BTP.

When solving P, MAC chooses a first variable x, assigns a value v to it and enforces AC. At this step, if no domain is empty, we know that a solution exists with the current assignment and MAC will find it in polynomial time like described in the proof of Theorem 7.6 of [5]. If a domain becomes empty, MAC considers the negative decision $x \neq v$ and again enforces AC. If there is no empty domain, MAC will find a solution in polynomial time like previously. Otherwise, no solution exists and the search stops. So, MAC solves P in polynomial time.

We reason in similar way for RFL, except that we have to consider, in the worst case, all the values of the first chosen variable. □

B. BTP in the benchmarks

Now, we wonder whether some instances usually exploited as benchmarks for solver comparisons belong to the BTP class or to one of its hidden classes. With this aim in view, we consider 2,681 binary benchmarks of the CSP 2008 Competition[6] and some of the most classical filterings, namely AC, PIC, $maxRPC$, SAC, NIC and SPC. For each instance, we check if the original instance and the instances obtained by applying one of the considered filtering belong to BTP. Checking whether an instance belongs to BTP has been performed as described in the proof of Theorem 3.2 of [5].

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

Table I provides the number of instances which belong to BTP or to one of its considered hidden classes and the number of these instances which are consistent w.r.t. the corresponding filtering. We can note that without any transformation, only 12 instances are BTP. In contrast, thanks to the exploitation of hidden classes, we can establish that more instances belong to a tractable class. For instance, 550 benchmarks (among which 47 are SAC-consistent) belong to the BTP^{SAC} class. Of course, we can observe that the more powerful the filtering is, the larger the corresponding hidden class is. Hence, naturally, the largest numbers of benchmarks belonging to a hidden class are reached by NIC and SPC.

Table II gives the names of some instances which belong to BTP or to one of its hidden classes. First, we can note the diversity of these instances (academic, random or real-world instances). Then, the presented results also highlight the fact that BTP is a hybrid tractable class. Indeed, some instances belong to BTP thanks to their particular structure (i.e. their constraint graph is acyclic) like hanoi-3_ext or graph12-w0 while others like pigeons-20-ord are BTP due to their particular relations.

Finally, from the viewpoint of the solving, the belonging to the BTP class or to one of its hidden classes may explain the efficiency of most solvers on these benchmarks. Indeed, in [5] (Theorem 7.6), it has been shown that MAC can solve any BTP instance in polynomial time without any additional processing. This result can be easily extended to RFL and more generally to any algorithm which maintains at each node a level of consistency (based only on value deletion) at least as powerful as arc-consistency. As most solvers of the state of the art exploit such a level of consistency, they are able to solve BTP instances in polynomial time. It is the same for instances belonging to a hidden class of BTP as soon as the solver enforces the suitable consistency as pre-processing, except for the class BTP^{AC} for which no pre-processing is required as stated in Theorem 6. For example, MAC and RFL solve the large-80-sat_ext instance in a backtrack-free manner independently from the chosen variable heuristic.

V. Conclusion

In this paper, we have studied the notion of *hidden tractable class* in relation to the methods used in CSP solvers of the state of the art. If the concept of *Hidden Structure* has already been discussed in [17] with the notions of *backbones* and *backdoors*, we treat it differently here while introducing a framework that seems more general. To this aim, we have introduced a formal framework for defining the notion of *hidden class discoverable by transformations of instances*. This framework allows to cover the concept of Hidden Structure proposed in [17] but also helps to develop other approaches. In particular, we have studied the notion of discoverable tractable classes with filterings, filterings being special cases of transformations. Specifically,

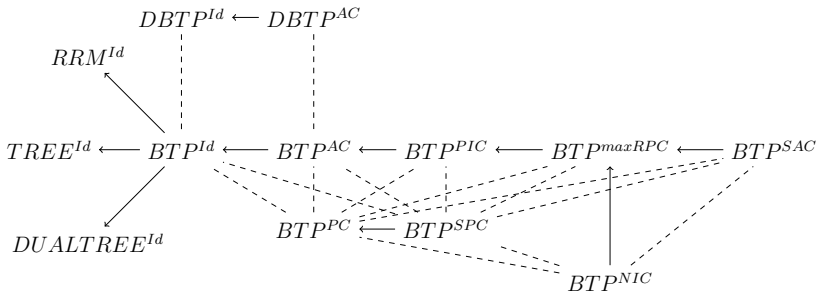

Figure 4. Relationship between classes around BTP.

Table I
NUMBER OF INSTANCES WHICH BELONG TO BTP OR TO ONE OF ITS HIDDEN CLASSES AND NUMBER OF THESE INSTANCES WHICH ARE CONSISTENT W.R.T. THE CORRESPONDING FILTERING.

	BTP	BTP^{AC}	BTP^{PIC}	BTP^{maxRPC}	BTP^{SAC}	BTP^{NIC}	BTP^{SPC}
# inst.	12	191	400	493	550	900	594
# cons.	-	46	47	47	47	83	71

Table II
SOME INSTANCES WHICH BELONG TO BTP OR TO ONE OF ITS HIDDEN CLASSES.

Instances	BTP	BTP^{AC}	BTP^{PIC}	BTP^{maxRPC}	BTP^{SAC}	BTP^{NIC}	BTP^{SPC}
bqwh-15-106-43_ext	no	no	no	no	no	no	yes
domino-100-100	no	yes	yes	yes	yes	yes	yes
ehi-90-315-96_ext	no	no	yes	yes	yes	yes	yes
ehi-90-315-97_ext	no	no	no	yes	yes	yes	yes
fapp17-0300-10	no	yes	yes	yes	yes	yes	yes
graph12-w0	yes	yes	yes	yes	yes	yes	yes
hanoi-3_ext	yes	yes	yes	yes	yes	yes	yes
langford-4-8	no	no	no	no	no	yes	yes
large-80-sat_ext	no	yes	yes	yes	yes	yes	yes
os-taillard-4-95-0	no	no	no	no	yes	yes	yes
pigeons-20-ord	yes	yes	yes	yes	yes	yes	yes
queens-4	no	no	yes	yes	yes	yes	yes
rand-23-23-253-131-48021_ext	no	no	no	no	no	yes	no
rand-2-40-180-84-900-93_ext	no	no	no	no	yes	no	yes
will199GPIA-6	no	no	no	no	no	yes	no

we have illustrated our approach on the class BTP [5]. From a practical viewpoint, we have shown that some instances among the benchmarks classically used by the community, belong to BTP after applying standard filterings (like AC, SAC, PIC, etc.).

However, our work is limited to the tractable class BTP and to some filterings. A natural way to extend this work would be to analyze other tractable classes, including classes defined for CSPs with constraints of arbitrary arity and their associated filterings. Another promising approach would be to proceed to a further analysis of the solving steps during search. Algorithms such as MAC or RFL operate by performing sequences of transformations of instances (variable assignments). So, it is possible that their efficiency is due to the fact that for some nodes of the search trees, once processed, the resulting instances belong to hidden tractable classes discoverable by these processing steps. Thanks to this analysis, it would be also possible to highlight new tractable classes based on the analysis of instances easily solved by standard algorithms, considering instances that seem to not belong explicitly to known tractable classes. Also, a formal study between backdoors and hidden tractable classes need also to be developed. Finally, the concept of transformation of instances was introduced here restrictively since it is essentially defined by simplifications of instances. It would be interesting to extend it, for example by considering transformations adding variables or values, and also using constraint relaxation. Such an analysis could perhaps allow to define new tractable classes which could be discoverable using these new transformations.

ACKNOWLEDGMENTS

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

REFERENCES

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

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

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

[4] D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of ECAI*, pages 125–129, 1994.

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

[6] C. Bessière, J.-C. Régin, R.H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185, 2005.

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

[8] R. Debruyne and C. Bessière. From restricted path consistency to max-restricted path consistency. In *Proceedings of CP*, pages 312–326, 1997.

[9] R. Debruyne and C. Bessière. Domain Filtering Consistencies. *JAIR*, 14:205–230, 2001.

[10] C. Bessière and R. Debruyne. Optimal and suboptimal singleton arc consistency algorithms. In *Proceedings of IJCAI*, pages 54–59, 2005.

[11] E. Freuder and C.D. Elfe. Neighborhood inverse consistency preprocessing. In *Proceedings of AAAI*, pages 202–208, 1996.

[12] R. Debruyne. A property of path inverse consistency for constraint satisfaction. In *Proceedings of ECAI*, pages 88–92, 2000.

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

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

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

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

[17] R. Williams, C. P. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of IJCAI*, pages 1173–1178, 2003.

[18] R. Dechter and J. Pearl. The Cycle-cutset method for Improving Search Performance in AI Applications. In *Proceedings of IEEE Conference on Artificial Intelligence Applications*, pages 224–230, 1987.

[19] CSPLib: A problem library for constraints. http://www.csplib.org.

[20] C. Lecoutre. *Constraint Networks - Techniques and Algorithms*. ISTE/Wiley, 2009.

[21] A. El Mouelhi, P. Jégou, and C. Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. In *Proceedings of ICTAI*, pages 947–954, 2013.