A New Filtering Based on Decomposition of Constraint Sub-Networks

Philippe Jégou and Cyril Terrioux
LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3)
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{philippe.jegou, cyril.terrioux}@univ-cezanne.fr

Abstract

In this paper, we introduce a new partial consistency for constraint networks which is called *Structural Consistency* of level w and is denoted w-*SC* consistency. This consistency is based on a new approach. While conventional consistencies generally rely on local properties extended to the entire network, this new partial consistency considers global consistency on subproblems. These subproblems are defined by partial constraint graphs whose tree-width is bounded by a constant w. We introduce a filtering algorithm which achieves w-SC consistency. We also analyze w-SC filtering w.r.t. other classical local consistencies to show that this consistency is generally incomparable. Finally, we present experimental results to assess the usefulness of this approach. We show that w-SC is a significantly more powerful level of filtering and more effective w.r.t. the runtime than SAC. We also show that w-SC is a complementary approach to AC or SAC. So we can offer a combination of filterings, whose power is greater than w-SC or SAC.

1 Introduction

It is well known that the CSP formalism (Constraint Satisfaction Problems [1]) is important in the field of AI to express and then to efficiently solve a large class of problems. A CSP, also called constraint network, consists in a set of variables X which must be assigned in their associated finite domains given by D, and a solution must satisfy a finite set C of constraints. Classical approaches to find solutions are based on backtracking algorithms whose time complexity cost is O($e.a.d^n$) where n is the number of variables, e is the number of constraints, a denotes a bound on constraint arity, and d is the maximum size of domains, assuming that the cost of a constraint check is O(a). To efficiently solve CSP, algorithms use generally filtering techniques, before search as preprocessing or during search. The quality of this basic tool is generally crucial for the efficiency of the search.

The effect of filtering techniques consists generally in removing values from domains. These values can be safely deleted because they cannot appear in solutions (they are not consistent). So, filterings are based on the notion of consistency. Because removing all inconsistent values is generally unrealistic from a practical viewpoint (it is an NP-hard problem), filterings are based on local consistencies which are relaxed consistency properties. So, a value can satisfy a local consistency, even if it does not appear in any solution. Nevertheless other values can contradict partial consistency and then be removed without modifying the satisfiability of a CSP. Partial consistencies [2] are generally defined by local consistencies which are extended to the whole constraint network. For example, to satisfy arc-consistency (AC), the most popular local consistency, a value needs to possess at least one compatible value (called a support) in the domain of its neighbouring variables. Otherwise, this value is removed from its domain (filtered) and this deletion can produce other deletions in the domains of its neighbouring variables. By using a mechanism called constraint propagation, the first deletion can finally be extended to the whole network. So, partial consistencies are generally local properties which must be verified on the whole network. From a practical viewpoint, the interest of a partial consistency is related to its filtering power and to the cost for enforcing it (time and space complexities).

In this paper, we introduce a new kind of consistency which must satisfy two criteria, particularly, on hard instances:

- practical efficiency,

- filtering power.

For that, this new consistency will be:

- parametrized, to control the complexity of the filtering,
- adjusted to the structure of the constraint network, by exploiting its substructures,
- adjusted to the tightness of the constraints in selecting the tightest constraints.

This new consistency is called w-*SC* because it is a parametrized Consistency (w is the parameter) based on Structural properties of the network. It is defined on a relaxation of the considered CSP (a subproblem) which is a partial constraint network whose tree-width is bounded by a constant w [3]. We choose this kind of subproblems because they can be solved in polynomial time (in w). Furthermore, thanks to recent progresses on decomposition methods, they can be managed really efficiently from a practical viewpoint [4]. Moreover, while classical partial consistencies consider constraints independently from their tightness, here we can select subsets of constraints focusing particularly on tight constraints in selecting the considered subgraph. More precisely, given a sub-network corresponding to a partial graph of bounded tree-width, we will say that a value satisfies w-SC consistency if it appears at least in one solution of the considered subproblem. It is based on the same kind of idea as known inverse consistencies but it is formally different. So, this new consistency allows us to define a new kind of filtering which is finally different from filterings generally used in CSPs [5]. Notably, we will show that w-SC is incomparable with existing efficient partial consistencies as AC or SAC [6]. Particularly, experiments show that the behavior of w-SC consistency is different from the behavior of these consistencies, being more efficient for hard instances, while being less efficient on easy (unconstrained instances). Moreover, we have observed that the values deleted by w-SC are generally not deleted by AC or SAC, and the reverse is true. Hence, w-SC can be considered as a local consistency which is a complementary consistency of already known consistencies. So we propose a combination of filterings based in SAC and w-SC whose power is significantly greater than w-SC or SAC. We also show that w-SC is a significantly more powerful level of filtering and more effective w.r.t. the runtime than SAC.

This paper is organized as follows. The next section recalls classical notions on partial consistencies and their related filterings. Then, in section 3, we introduce w-SC consistency and its associated filtering. Section 4 presents a theoretical analysis of relations between w-SC consistency and other consistencies, while section 5 provides an empirical analysis of this filtering. Finally, the last section is devoted to a discussion about future works.

2 Preliminaries

A *finite constraint satisfaction problem* or *finite constraint network* (X, D, C) is defined as a set of variables $X = \{x_1, \dots x_n\}$, a set of domains $D = \{D(x_1), \dots D(x_n)\}$ (the domain $D(x_i)$ contains the possible values for the variable x_i), and a set C of constraints among variables. A constraint $c_i \in C$ is defined by its scope, denoted $S_C(c_i)$ and by an associated relation $R_C(c_i)$. The scope is an ordered subset of variables, that is $S_C(c_i) = (x_{i_1}, x_{i_2}, \dots x_{i_{a_i}})$ where a_i is called the *arity* of the constraint c_i. The relation $R_C(c_i) \subseteq D(x_{i_1}) \times D(x_{i_2}) \dots \times D(x_{i_{a_i}})$ defines the allowed combinations of values for the variables in $S_C(c_i)$. Here, we denote by S_C the set of scopes of the constraints, that is $S_C = \{Sc_1, Sc_2 \dots Sc_e\}$ where $e = |C|$ is the number of constraints. A solution of (X, D, C) is an assignment of each variable which satisfies all the constraints. Without loss of generality, we assume that each variable is involved in at least one constraint. If every constraint of a CSP is binary (i.e. involves exactly two variables), then the structure of this binary network (called a binary CSP) can be represented by the graph (X, S_C) called the *constraint graph*.

In this paper, we assume that the relations are not empty. Moreover, without loss of generality, we will assume that the constraint network is connected and normalized (two different constraints do not involve the same variables) and for lack of space, we will consider here only binary networks. So, for a constraint c_k such $S_C(c_k) = (x_i, x_j)$, c_k will be denoted c_{ij}. Generally, CSPs are solved using backtracking algorithms which can be really efficient if they efficiently exploit filterings before or during search to avoid redundant search. These filterings are formally based on the notion of local consistency.

The most popular and oldest local consistency is called *arc-consistency* (AC). Given a CSP $P = (X, D, C)$, a value $v_i \in D(x_i)$ is arc-consistent with $c_{ij} \in C$ iff there exists a valid value $v_j \in D(x_j)$ s.t. $(v_i, v_j) \in R_C(c_{ij})$. 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 on c_{ij} iff $\forall v_i \in D(x_i)$,

the value v_i is arc-consistent, and the CSP $P = (X, D, C)$ is arc-consistent iff $\forall D(x_i) \in D$, the domain $D(x_i)$ is arc-consistent with all $c_{ij} \in C$. A filtering of domains based on AC consists in removing the values which do not satisfy arc-consistency. When a value v_i is removed, a mechanism called *constraint propagation* can be run to remove values which were supported only by v_i (no other value of $D(x_i)$ is compatible with them), and this process can be extended to other values. For binary networks, AC-2001 [7] 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. [8] has introduced *k-consistency* which considers subsets of k variables. For k-consistency, a new constraint (its arity is $k-1$) is added to the network when a consistent assignment on $k-1$ variables cannot be extended to a k^{th} variable. If the network is i-consistent, for $2 \leq i \leq k$, the CSP is said *strong k-consistent*. The greater the value of k is, the more powerful the filtering is. Unfortunately, the time and space complexity is O($n^k.d^k$) [9] and then this kind of filtering has important drawbacks. Because of the time and space complexity, these filterings are generally unusable even for small values of k ($k = 3$ is frequently unrealistic for practical cases). Moreover, the filtering will add new constraints in the network, their arity being $k - 1$ and then the necessary space can be prohibitive, even for small values of k. To avoid these problems, mainly the second one related to added constraints, other local consistencies have been introduced. For example, [5] have proposed the *k-inverse consistency*. The associated filtering removes values which cannot be extended to any $k - 1$ additional variables to form a consistent assignment. This filtering is more powerful than AC and it avoids the problem related to space complexity but for time complexity, the problem remains the same since it is O($n^k.d^k$). So, they have suggested to limit k-inverse consistency to small values of k. In [5], another kind of inverse consistency has been introduced which is defined in the same spirit and which is called *NIC* for *neighborhood-inverse consistency*. Here, 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 complexity is related to the maximum degree Δ of a variable in the constraint network, and then the time complexity of the proposed algorithm is O($\Delta^2.(n+e.d).d^{\Delta+1}$). 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-consistency* (denoted *SAC*) [6] 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 arc-consistent sub-domains. The time complexity is O($e.n.d^3$) [10].
To conclude this overview, we must recall that from a practical viewpoint, the local consistencies generally considered as the most usable in practice are AC or SAC which seem to obtain the best compromise between the time cost (and its practical efficiency) and its filtering power.

3 Structural Consistency

3.1 w-SC Consistency

Structural consistency is based on the notion of partial graph whose *tree-width* is bounded by a constant w. The tree-width is based on the notion of *tree-decomposition* which has been formally introduced in [3]. It has been exploited in the field of CSP to define tractable classes [11] and to propose efficient methods for solving constraint networks which possess good topological properties [12, 4].
Our objective here is then different from these works since we will use it to define local consistencies. For that, we introduce the notion of w-*PST* which corresponds to *partial spanning tree-decomposition* of tree-width w. Before, we recall the classical notion of tree-decomposition:

Definition 1 *A tree-decomposition of a graph $G = (X, E)$ is a pair (N, T) where $T = (I, F)$ is a tree with nodes I and edges F and $N = \{N_i : i \in I\}$ is a family of subsets of X, s.t. each subset (called cluster) N_i is a node of T and verifies:*

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

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

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

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

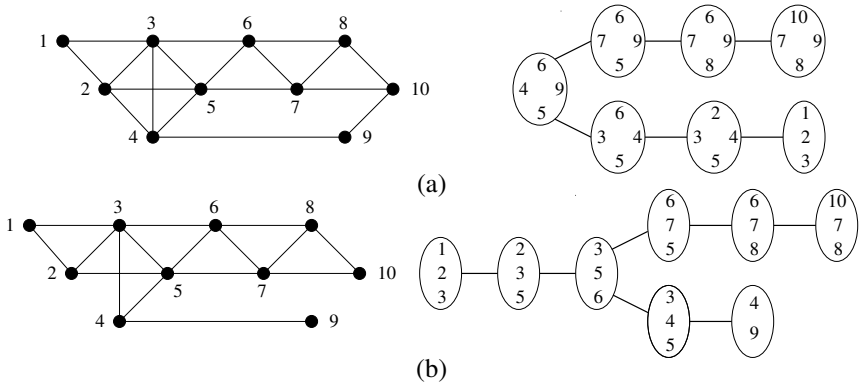


Figure 1: (a) A graph of tree-width 3 and the corresponding tree-decomposition. (b) A 2-PST of the graph given in (a), and one tree-decomposition.

Now, we define partial graphs with particular tree-width.

Definition 2 *Given a graph $G = (V, E)$, a partial graph of G is a subgraph $G' = (V, E')$ where $E' \subseteq E$. A partial spanning tree-decomposition of tree-width w for G, denoted w-PST, is a partial graph of G whose tree-width is w.*

The graph given in figure 1(a) is a 3-PST because its tree-width is 3 (an optimal tree-decomposition is given in this figure). In figure 1(b), we have the partial graph induced by the deletion of edges $\{2, 4\}$ and $\{9, 10\}$ in the graph given in figure 1(a). It is a 2-PST because its tree-width is 2 as indicated by the optimal tree-decomposition given in this figure.
Now, we introduce the notion of subproblem of a given CSP induced by the assignment of a variable.

Definition 3 *Given a CSP $P = (X, D, C)$, a variable $x_i \in X$ and a value $v_i \in D(x_i)$, the subproblem of P induced by the assignment $x_i = v_i$ is $P|_{x_i=v_i} = (X, D', C')$ where $D'(x_i) = \{v_i\}$ and for all $j \neq i, D'(x_j) = D(x_j)$ and $C' = C$ except for the relations associated to the constraints including x_i which are restricted to the tuples where the value v_i appears.*

We introduce now the notion of relaxed problem of a given CSP which is defined by a subset of constraints.

Definition 4 *Given a CSP $P = (X, D, C)$ and $W \subseteq S_C$, the relaxed problem of P induced by W is $P(W) = (X, D, C')$ where $W = S_{C'}$.*

For w-SC, the relaxed problem is defined by a subset of constraints forming a partial spanning tree-decomposition of tree-width w.

Definition 5 *Given a CSP $P = (X, D, C)$ and a w-PST $G = (X, W)$ of (X, S_C):*

- *The value $v_i \in D(x_i)$ is w-SC-consistent w.r.t. G iff $P(W)|_{x_i=v_i}$ has a solution.*
- *The domain $D(x_i)$ is w-SC-consistent w.r.t. G iff $\forall v_i \in D(x_i)$, the value v_i is w-SC-consistent w.r.t. G.*
- *The CSP $P = (X, D, C)$ is w-SC-consistent w.r.t. G iff $\forall D(x_i) \in D$, $D(x_i)$ is w-SC-consistent w.r.t. G.*

Note that for a given CSP and a given value w, there is different possible consistencies, since one consistency is defined with respect to one particular w-PST. Moreover, this definition of consistency is related to values of domains but it can be easily generalized to partial assignments of subsets of variables. For lack of space and to simplify our presentation, we limit here w-SC-consistency to values.

3.2 Filtering

As classically for consistencies, the filtering associated to w-SC consistency consists in deleting values which do not satisfy it.

Definition 6 *Given a CSP $P = (X, D, C)$ and a w-PST $G = (X, W)$ of (X, S_C), the filtered CSP using w-SC-consistency is w-$SC(P, W) = (X, D', C')$ where:*

- $D' = \{D'(x_1), \dots D'(x_n)\}$ *where* $\forall D'(x_i) \in D'$, $D'(x_i) = \{v_i \in D(x_i)$*: v_i is w-SC-consistent w.r.t. G*$\}$.
- $S_{C'} = S_C$.
- $\forall c'_{ij} \in C', R_{C'}(c'_{ij}) = R_C(c_{ij}) \cap D'(x_i) \times D'(x_j)$.

Note that given a CSP and a w-PST (X, W) of (X, S_C), w-$SC(P, W)$ is unique. Moreover, to ensure that w-SC consistency defines a valid filtering, we must also ensure that no filtered value can appear in solutions of the given CSP. It is necessarily the case since removed values cannot appear in solutions of a relaxed CSP. Now, we present the algorithm called *Comp-w-SC* which achieves w-SC filtering. Contrary to classical filtering algorithms as those enforcing AC, this consistency does not need propagation after deletions. Indeed, while classical algorithms remove values and propagate these deletions, here once a value v_i is validated finding a solution, it will not be deleted after, and thus v_i is definitively validated. It is because v_i appears in a solution of the relaxed CSP, and because the other values that appear in this solution (which can be considered as supports for the value v_i) are also validated by the same reason. Thus, since these values are also validated, it will not be necessary to check their w-SC consistency.

In *Comp-w-SC*, the function *Solution($P(W)|_{x_i=v_i}$, Sol)* is called to check consistency. If v_i appears in a solution $Sol = (v_1, v_2, \dots v_i, \dots v_n)$ of $P(W)$, the function returns *true* and Sol is the other result of this call. Otherwise, it returns *false*. In *Comp-w-SC*, D' corresponds to the set of domains containing values which have already been validated and then memorized during the filtering. So, if a value v_i already appears in $D'(x_i)$ it is because this value already appears in a solution and then it will not be necessary to check it after for its w-SC consistency. Note that at the end of *Comp-w-SC*, for all variable x_i, we have $D(x_i) = D'(x_i)$.

Algorithm 1: Comp-w-SC(**In**: (X, W): Graph;**InOut**: $P = (X, D, C)$: CSP)

```
for x_i ∈ X do
    D'(x_i) ← ∅;
end
for x_i ∈ X do
    for v_i ∈ D(x_i) do
        if v_i ∉ D'(x_i) then
            if Solution(P(W)|_{x_i=v_i}, Sol) then
                for v_j ∈ Sol do
                    D'(x_j) ← D'(x_j) ∪ {v_j}
                end
            else
                D(x_i) ← D(x_i) − {v_i}
            end
        end
    end
end
```

Property 1 *The time complexity of Comp-k-SC is $O(n^2.w.d^{w+2})$ while its space complexity is $O(n.w.d^w)$.*

Proof: The function *Solution($P(W)|_{x_i=v_i}$)* is called at most $n.d$ times and the cost of one call to this function is bounded by $n.w.d^{w+1}$. Indeed, it can be implemented using algorithms based on tree-decomposition of CSPs such as TC [12] or BTD [4].

Moreover, we know that the space complexity of decomposition methods as BTD is related to the size of the separators between cliques [4]. Here, the maximum size of separators in the w-PST is w and their number is at most $n - 1$. So, the space complexity is bounded by O($n.w.d^w$). □

4 Relations with other consistencies

To evaluate the power of the w-SC filtering, we provide here an analysis in the same spirit as in [6] which presents the comparison between numerous partial consistencies. Here, we consider AC, SAC, strong-PC, and more generally strong-k-consistency and k-inverse consistency. 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.

Note that for a given CSP, and a given value w, the number of possible filterings is potentially related to the number of possible w-PSTs. Nevertheless, we can easily find instances of CSPs such that next properties hold.

Theorem 1 *1-SC $<$ AC and for $w > 1$, w-SC and AC are incomparable.*

Proof: It is clear that connected 1-PST are exactly trees. So, since in a tree, a value appears in a solution iff it verifies AC, 1-SC cannot filter more values than the arc-consistency, which considers the whole problem, does. Thus, we have 1-SC $\leq$ AC. Moreover, since other values of a network can be deleted by AC, exploiting constraints that does not appear in the considered 1-PST, we have also 1-SC $<$ AC.

Now, if we consider w-SC and AC for $w > 1$, they are incomparable. Indeed, it is sufficient to see that if a value in the domain of a variable is removed because it has no support for a constraint which does not appear in a w-PST, then this value can be conserved by w-SC filtering. Conversely, a value can be removed by w-SC filtering but not by AC. □

Theorem 2 *For $w > 1$, w-SC and SAC are incomparable.*

Proof: Since 1-SC $<$ AC and since AC $<$ SAC, by transitivity of $<$, we have 1-SC $<$ SAC. Now, since SAC considers all the constraints which appear in the network, necessarily, the filtering can delete values that will not be deleted by 2-SC. Conversely, if we consider the example (c) given in page 216 of [6] which is a 2-PST satisfying SAC, we can easily see that 2-SC will remove the value deleted by strong-PC while this value is not deleted by SAC. Thus, 2-SC and SAC are incomparable. □

Before comparing stronger consistencies, we define relations between different levels of w-SC consistency. For relations between w-SC with different values of w, we assume that we consider CSPs which possess partial spanning tree-decompositions with different widths such that w-PST $\subseteq (w+1)$-PST (edges of the considered w-PST satisfy this condition). Otherwise we have no guarantee about the comparison between w-SC and $(w+1)$-SC.

Theorem 3 *If w-PST $\subseteq (w+1)$-PST, then w-SC $<$ $(w+1)$-SC.*

Proof: If w-PST $\subseteq (w+1)$-PST, the values removed by w-SC consistency considering the w-PST are necessary removed considering the $(w+1)$-PST. Moreover, since there is constraints in the $(w+1)$-PST that do not appear in the w-PST, additional values will be removed by $(w+1)$-SC consistency in considering the $(w+1)$-PST. Thus, we have w-SC $<$ $(w+1)$-SC. □

More generally, applying the same principle as for the theorem 1, we have the next property that can be considered as its generalization:

Theorem 4 *k-SC $<$ strong-(k+1)-consistency and for $k > 2$, k-SC and strong-k-consistency are incomparable.*

Proof: Firstly, we show that k-SC $<$ strong-$(k+1)$-consistency. It is easy to see that the width [13] of a w-PST is exactly w. Consequently, applying the results proposed in [13] which links the width of a constraint network to the level of strong-consistency that it verifies, necessarily, every value that appears in a domain satisfying strong-$(k+1)$-consistency belongs to a solution of this k-PST. Thus, it will not be deleted by k-SC consistency. Moreover,

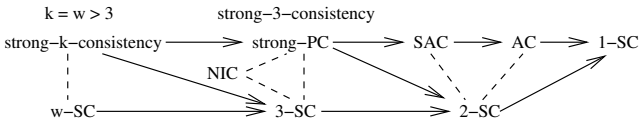


Figure 2: Relations between consistencies

since other values of the network can be deleted by strong-$(k+1)$-consistency, exploiting constraints that do not appear in the considered w-PST (with $w = k$), it is easy to see that strong-$(k+1)$-consistency can delete more values than k-SC consistency filtering does, and consequently, we have k-SC $<$ strong-$(k+1)$-consistency.
To show that for $k > 2$, k-SC and strong-k-consistency are incomparable, it is then sufficient to show that some values can be deleted by k-SC, while they are not filtered by strong-k-consistency. Consider the CSP corresponding to the k-coloring problem on the complete graph on $k+1$ vertices. This network is a k-PST but it does not admit a solution. Thus, achieving of k-SC consistency on this network will delete all values. Now, if we consider strong-k-consistency, no value will be deleted. Consequently, for $k > 2$, k-SC and strong-k-consistency are incomparable. □

So, we have the next trivial corollary:

Corollary 1 *2-SC $<$ strong-PC.*

Finally, note that we can replace strong-k-consistency by k-inverse-consistency [5]. It is also easy to establish that NIC and k-SC (for $k > 2$) are incomparable.
In figure 2, relations between consistencies are summarized. Arrows represents the relation $>$ while dotted lines indicates that two consistencies are incomparable.

5 Experiments

5.1 Experimental protocol

The quality of the filtering performed by w-SC mainly depends on the considered w-PST. The computation of the w-PST is achieved thanks to a heuristic method related to the notion of k-tree [14]. We exploit k-trees here because they are the graphs which, for a given tree-width, have the maximum number of edges, and thus can potentially contain the maximum number of constraints. A graph G is a k-tree if G has k vertices and is a complete graph (it is a trivial k-tree) or there is a vertex x of degree k whose neighborhood induces a complete graph and the graph obtained by removing x and all edges incident to it from G is a k-tree. Then, k-trees can be built starting with a complete graph of k vertices, and each time a vertex x is added, connecting it to vertices of a k-clique in the previous graph. In our case, we choose at each step the vertex x which has the smallest value $\prod_{c_{ij}} t_{ij}$ where c_{ij} is a constraint spanned by the clique formed by x and the vertices of the considered k-clique in the previous graph and t_{ij} is the ratio of the number of forbidden tuples over the number of possibles tuples. Likewise the initial clique is chosen in a similar way. By so doing, we aim to compute tight subproblems and so obtain a more powerful filtering. We also implement another method which computes first a k-tree as previously described and then tries heuristically to add as many constraints (among the constraints which do not belong to the k-tree yet) as possible such that the tree-width remains bounded by w. The remaining constraints are processed in the decreasing tightness order. In the following results, w-SC1 (respectively w-SC2) denotes the application of our algorithm with a w-PST computed thanks to the first method (resp. thanks to the second one). In both cases, we exploit a k-tree with $k = w$ and the subproblem related to the considered w-PST is solved thanks to BTD [4].
Regarding AC and SAC, we have implemented AC2001 [7] and a naive version of SAC. All the algorithms are written in C. Note that we have also compared our results with a clever implementation of SAC (namely SAC3 [15] provided in the Java solver Abscon). Regarding the runtime, our implementation performs sometimes worse than SAC3 but the results of the comparison with w-SC remain the same. Of course, both versions of SAC have the same filtering power. So, in order to make easier the implementation of the combination of SAC and w-SC

Classes	AC			SAC			6-SC1			6-SC2		
(n,d,e,t)	time	#inc	#rv	time	#inc	#rv	time	#inc	#rv	time	#inc	#rv
(100,20,495,275)	1.8	0	9.24	198	50	104.68	70	20	486.82	441	48	79.20
(100,20,990,220)	2.4	0	0.22	11987	11	92.04	105	21	566.08	240	48	79.32
(100,20,1485,190)	3.4	0	0	4207	0	0.40	286	31	494.40	187	49	38.30
(100,40,495,1230)	4.6	0	0.92	3239	50	345.06	270	21	621.94	5709	48	106.76
(100,40,990,1030)	5.8	0	0	13229	0	0.08	515	30	809.34	4954	48	176.42
(100,40,1485,899)	8.2	0	0	11166	0	0	1622	32	902.14	1854	48	181.18
(200,10,1990,49)	2.6	0	20.96	88	50	38.28	128	22	350.62	72	48	56.36
(200,10,3980,35)	5.8	0	0.92	10503	49	261.74	248	0	34.86	637	0	249.56
(200,10,5970,30)	7.8	0	0.24	11335	0	11	423	0	54.62	708	2	241.34
(200,20,995,290)	4.6	0	57.58	224	50	65.52	190	26	670.32	7464	49	78.42
(200,20,1990,245)	6	0	3.36	3716	50	256.62	192	32	592.96	1109	50	20
(200,20,3980,195)	12.4	0	0.04	34871	0	1.82	573	25	808.46	592	49	70.48
(200,20,5970,165)	17	0	0	23307	0	0.04	2242	10	1179.88	1600	43	280.3

Table 1: Runtime (in ms), number of instances detected as inconsistent and mean number of removed values. All the considered instances have no solution.

(see subsection 5.3), we only consider our naive implementation in the provided results.
These algorithms are compared on random instances produced by the random generator written by D. Frost, C. Bessière, R. Dechter and J.-C. Régin. Note that we do not use here structured random instances because, even if w-SC takes benefit from the underlying structure of the CSP, it aims to be run on general CSP instances as most of filtering algorithms. This generator takes 4 parameters n, d, e and T. It builds a CSP of class (n, d, e, t) with n variables which have domains of size d and e binary constraints ($0 \leq e \leq \frac{n(n-1)}{2}$) in which t tuples are forbidden ($0 \leq t \leq d^2$). The presented results are the averages of the results obtained on 50 instances (with a connected constraint graph) per class (except for figures 4 and 5 where we consider only 30 instances per class). The experimentations are performed on a linux-based PC with an Intel Pentium IV 3.2 GHz and 1 GB of memory.

5.2 AC/SAC vs w-SC

We have tested many classes of instances by varying the number of variables, the size of domains (up to 40 values), the constraint graph density and the tightness. Here, we only provide:

- the results obtained by varying t for 200 variables, 20 values per domain and 5,970 constraints (density of 30%) in figure 3 or 19,900 constraints (complete graph) in figure 5,
- the results obtained by varying t for 100 variables, 40 values per domain and 990 constraints (density of 20%) in figure 4,
- the results on some representative classes in table 1 (we observed similar results with higher or lesser densities).

Note that we do not provide the results of w-SC2 in figure 3 because the results are very close to ones of w-SC1 (w-SC2 only detects the inconsistency of a few additional instances while it spends a slightly greater time).
Before comparing our algorithm with AC or SAC, we raise the question of the choice of a good value for w. In figure 3, if we consider the number of instances which are detected as inconsistent, we can note that this number for w-SC increases as the value of w increases. Such a result is foreseeable since for larger values of w, w-SC takes into account more constraints and is able to perform a more powerful filtering. The same result generally holds for the runtime which increases with w. According to our observations, the value 6 for w seems to correspond to the best trade-off between the runtime and the power of the filtering. On the one hand, by exploiting 6-PSTs, 6-SC1 (or 6-SC2) takes into account enough constraints in order to enforce an efficient filtering. On the other hand, with larger values of w, the number of removed values and so the number of detected inconsistent instances are not significantly improved while the runtime and the space requirement may increase significantly w.r.t. $w = 6$.

Class (n, d, e, t)	AC	SAC	6-SC1	AC $\cap$ 6-SC1	SAC $\cap$ 6-SC1
(100,40,990,970)	0	0	186.49	0	0
(100,40,1485,860)	0	0	786.52	0	0
(200,10,1990,40)	2.38	15.9	0.86	0.48	0.78
(200,10,3980,30)	0.14	0.58	0.04	0.04	0.04
(200,10,5970,30)	0.24	11	54.62	0.02	0.76
(200,20,1990,230)	0.74	33.4	138.1	0.16	5.24
(200,20,14925,150)	0	0.08	1154.92	0	0.06

Table 2: Mean number of values deleted by the different algorithms and mean number of values which are both removed by AC and 6-SC1 or SAC and 6-SC1. All the considered instances have no solution but have been found consistent by both AC, SAC and 6-SC1.

Then, if we compare w-SC1 and w-SC2, we can observe in table 1 that generally w-SC2 detects more instances as inconsistent than w-SC1. Again, such a result was foreseeable, since w-SC2 takes into account more constraints than w-SC1. For the same reason, w-SC2 often spends more time for achieving SC

Now, if we compare the w-SC filtering with AC or SAC w.r.t the number of instances which are detected as inconsistent, we can note, in figure 3, that 3-SC detects more inconsistencies than AC while SAC may perform better or worse than w-SC depending on the value of w. Nevertheless, we can observe that 6-SC often detects more instances as inconsistent than SAC. On the 650 instances considered in table 1, we observe that, SAC performs often better than 6-SC1 (310 instances are detected as inconsistent by SAC against 270 by 6-SC1) but worse than 6-SC2 (310 against 530).

Regarding the runtime, according to figure 3(a), AC is generally faster than w-SC, except when the instances are obviously inconsistent. In such a case, w-SC often succeeds in detecting the inconsistency by removing all the values of the first considered variable while AC needs many deletions. Compared with SAC, w-SC spends more time than SAC only on instances which are not tight enough (for $t < 50$ in figure 3(a)) and so obviously consistent, because many calls to BTD are required. In contrast, when the tightness is closer to the consistent/inconsistent threshold or above, w-SC performs faster than SAC. Indeed, w-SC removes less values than SAC in order to detect the inconsistency. The main reason is that, by construction, it checks (and possibly removes) each value of a variable before considering a new variable while in AC or SAC, the values of a given variable are generally deleted at different (and non-consecutive) moments (due to the propagation mechanism). Finally, we have observed that the behavior of w-SC with respect to AC or SAC is improved when the density of the constraint graph increases. So, w-SC succeeds in outperforming significantly SAC for time efficiency and AC and SAC for detection of inconsistencies in the consistent/inconsistent threshold area.

5.3 Complementarity and combinations of AC/SAC and w-SC

If we compare the values which are deleted by the different algorithms on consistent instances, we can note that a value deleted by w-SC is not necessarily removed by AC or SAC and conversely. Table 2 illustrates perfectly this phenomenon. Such a report highlights the difference which exists between our new filtering and classical ones like AC or SAC and leads us to study the complementarity of w-SC with AC or SAC. With this aim in view, we combine here w-SC with AC or SAC. More precisely, from two filtering algorithms X and Y, we derive a new filtering algorithm denoted X+Y which consists in applying X and then, if the instance is not detected as inconsistent yet, Y. Table 3 provides the results obtained by AC+6-SC, SAC+6-SC, 6-SC+AC and 6-SC+SAC.

We can note that AC+6-SC slightly improves the detection of inconsistent instances w.r.t 6-SC (e.g. 275 instances are detected as inconsistent by AC+6-SC1 against 270 for 6-SC1) while 6-SC+AC performs really better with 450 instances found inconsistent by 6-SC1+AC (respectively 567 by 6-SC2+AC against 530 for 6-SC2). Both algorithms have better results than AC which do not succeed in detecting the inconsistency of any instance. Likewise, SAC+6-SC improves both the behaviour of SAC and 6-SC (457 and 587 instances respectively for SAC+6-SC1 and SAC+6-SC2 against 310 for SAC). For these three algorithms, the runtime does not exceed the cumulative runtime of AC and 6-SC or SAC and 6-SC. The more interesting results are provided by 6-SC+SAC. On the one hand, 6-SC1+SAC and 6-SC2+SAC detect respectively 598 and 640 instances as inconsistent (i.e. 92% and 98%

Classes	AC+6-SC1		SAC+6-SC1		6-SC1+AC		6-SC1+SAC		AC+6-SC2		SAC+6-SC2		6-SC2+AC		6-SC2+SAC	
(n,d,e,t)	time	#inc	time	#inc	time	#inc	time	#inc	time	#inc	time	#inc	time	#inc	time	#inc
(100,20,495,275)	70	20	197	50	71	41	86	50	381	48	198	50	386	50	430	50
(100,20,990,220)	104	21	12020	28	100	40	1387	49	238	48	12035	48	240	50	223	50
(100,20,1485,190)	272	31	4487	32	288	47	525	48	203	49	4385	49	202	50	185	50
(100,40,495,1230)	248	21	3249	50	268	30	608	50	5724	48	3240	50	4950	49	5754	50
(100,40,990,1030)	526	30	13835	30	579	41	1729	48	4982	48	18407	48	5546	50	5590	50
(100,40,1485,899)	1629	32	12822	32	1631	44	2466	48	1858	48	13015	48	1826	50	2024	50
(200,10,1990,49)	125	25	89	50	133	45	136	50	67	50	91	50	73	50	73	50
(200,10,3980,35)	261	0	10715	49	260	0	6946	49	654	0	10536	49	638	10	2701	50
(200,10,5970,30)	424	0	15939	0	424	1	15744	18	711	2	16060	3	728	12	5906	42
(200,20,995,290)	175	28	225	50	199	45	191	50	5215	50	223	50	7933	50	7965	50
(200,20,1990,245)	186	32	3774	50	181	45	352	50	1086	50	3614	50	1114	50	1133	50
(200,20,3980,195)	578	25	35291	26	539	42	3130	49	627	49	35352	49	593	50	549	50
(200,20,5970,165)	2253	10	25526	10	2235	29	15216	39	1696	43	24923	43	1592	46	5056	48

Table 3: Runtime (in ms) and number of instances detected as inconsistent for the combined algorithms.

of the considered instances against 48% for SAC, 42% for 6-SC1 and 82% for 6-SC2). On the other hand, its runtime is often significantly better than the cumulative runtime of SAC and 6-SC or than the runtime of SAC. These results clearly show that the filtering performed by AC/SAC and SC are not only significantly different but also complementary.

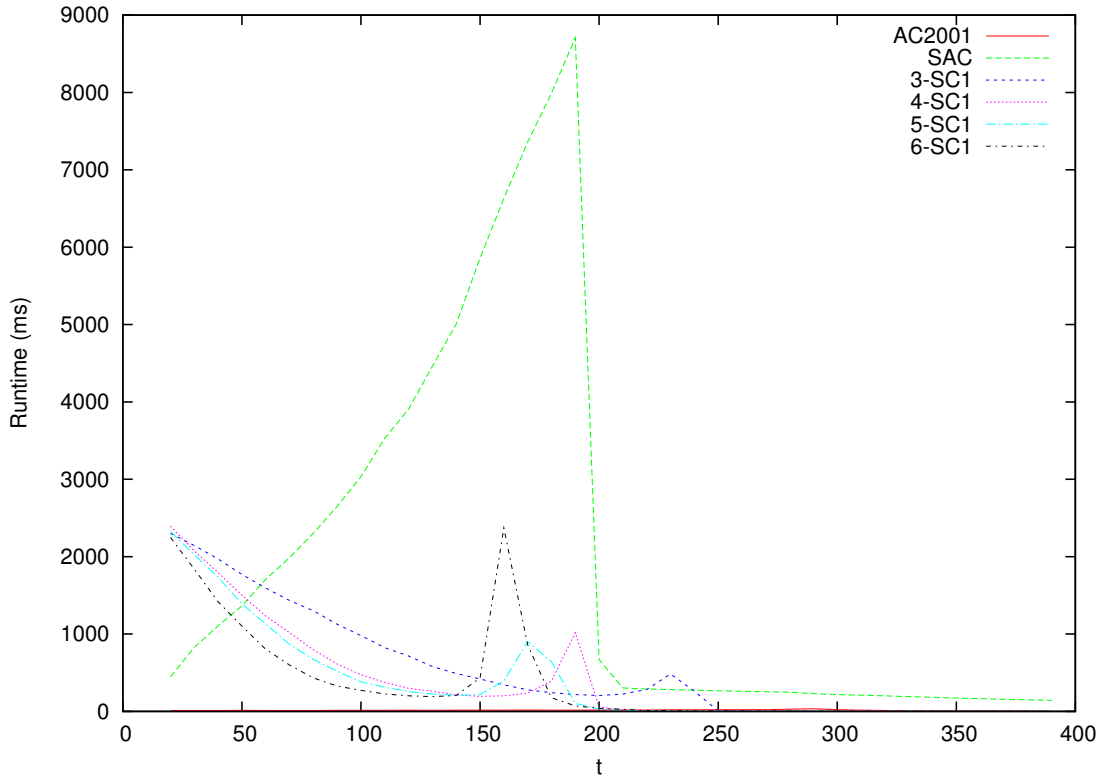
6 Discussion and conclusion

We have proposed a new parameterized partial consistency which allows to exploit the underlying properties of a constraint network, both at the structural level but also at the level of the tightness of constraints. This new partial consistency is different in its approach to those proposed previously. This can be seen in the theoretical comparisons we have provided, but also in the experiments since its filtering capabilities are different from those of conventional methods: this is not the same values that are removed, and time efficiency is not located in the same areas as other filterings. It is therefore a potentially complementary approach to existing ones. In experimental results, we show that w-SC offers a significantly more powerful level of filtering and more effective w.r.t the runtime than SAC. We also show that w-SC is a complementary approach to AC or SAC. So we can offer a combination of filterings, whose power is greater than w-SC or SAC.

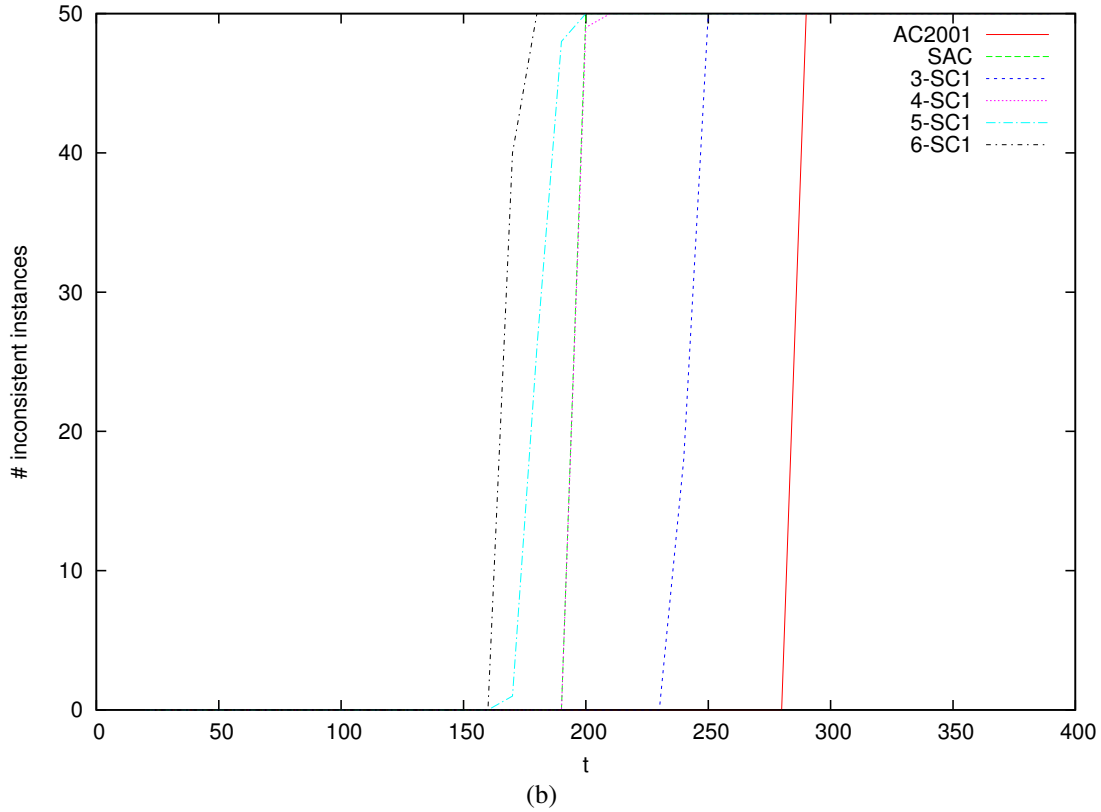
This leads naturally to study hybrid consistencies combining w-SC with other consistencies such as AC or SAC, to provide filtering both more robust and more powerful. Among the potential extensions of this work, w-SC could be extended to n-ary constraints, which does not seem to be technically difficult. We could also extend w-SC in extending filterings to partial assignments, for example by offering the concept of k-w-SC consistency which would produce new constraints whose arity is k. Moreover, even if 6-SC is faster and performs a more powerful filtering than SAC, it would also be necessary to better identify the good values of w. A natural track would be to assess the density of the w-PST for a given value of w. Another important way could be to use w-SC during the search. We should study both conventional methods based on backtracking, but also to improve the decomposition methods. Finally, a comprehensive study should be conducted based on the general framework proposed in [16] by studying different kinds of subproblems, not only related to partial spanning tree-decompositions. Nevertheless, this study could be of a limited interest. Indeed, this framework allows to define a generic algorithm which is based on a classic propagation architecture. Given a deleted value, this value will be propagated to delete other values. Conversely, achieving w-SC is not based on deletions propagation but runs using validations of value. This difference makes the use of the framework of [16] of a limited interest here for filtering

Bibliography

1. F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.
2. C. Bessiere. *Constraint Propagation*, chapter 3, pages 29–83. Handbook of Constraint Programming, F. Rossi, P. van Beek, T. Walsh, Elsevier, 2006.
3. N. Robertson and P.D. Seymour. Graph minors II: Algorithmic aspects of treewidth. *Algorithms*, 7:309–322, 1986.
4. P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.
5. E. Freuder and C.D. Elfe. Neighborhood inverse consistency preprocessing. In *Proceedings of AAAI*, pages 202–208, Portland, OR, USA, 1996.
6. R. Debruyne and C. Bessière. Domain Filtering Consistencies. *JAIR*, 14:205–230, 2001.
7. C. Bessière, JC. Régin, R.H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185, 2005.
8. E. Freuder. Synthesizing constraint expressions. *CACM*, 21(11):958–966, 1978.
9. M.C Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41(1):89–95, 1989.
10. C. Bessière and R. Debruyne. Optimal and suboptimal singleton arc consistency algorithms. In *Proceedings of IJCAI*, pages 54–59, 2005.
11. G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:343–282, 2000.
12. R. Dechter. *Constraint processing*. Morgan Kaufmann Publishers, 2003.
13. E. Freuder. A Sufficient Condition for Backtrack-Free Search. *JACM*, 29 (1):24–32, 1982.
14. Beineke and Pippert. Properties and characterizations of k-trees. *Mathematika*, 18:141–151, 1971.
15. C. Bessière, S. Cardon, R. Debruyne, and C. Lecoutre. Efficient Algorithms for Singleton Arc Consistency. *Constraints*, 2010. To appear.
16. G. Verfaillie, D. Martinez, and C. Bessière. A Generic Customizable Framework for Inverse Local Consistency. In *Proceedings of AAAI*, pages 169–174, 1999.

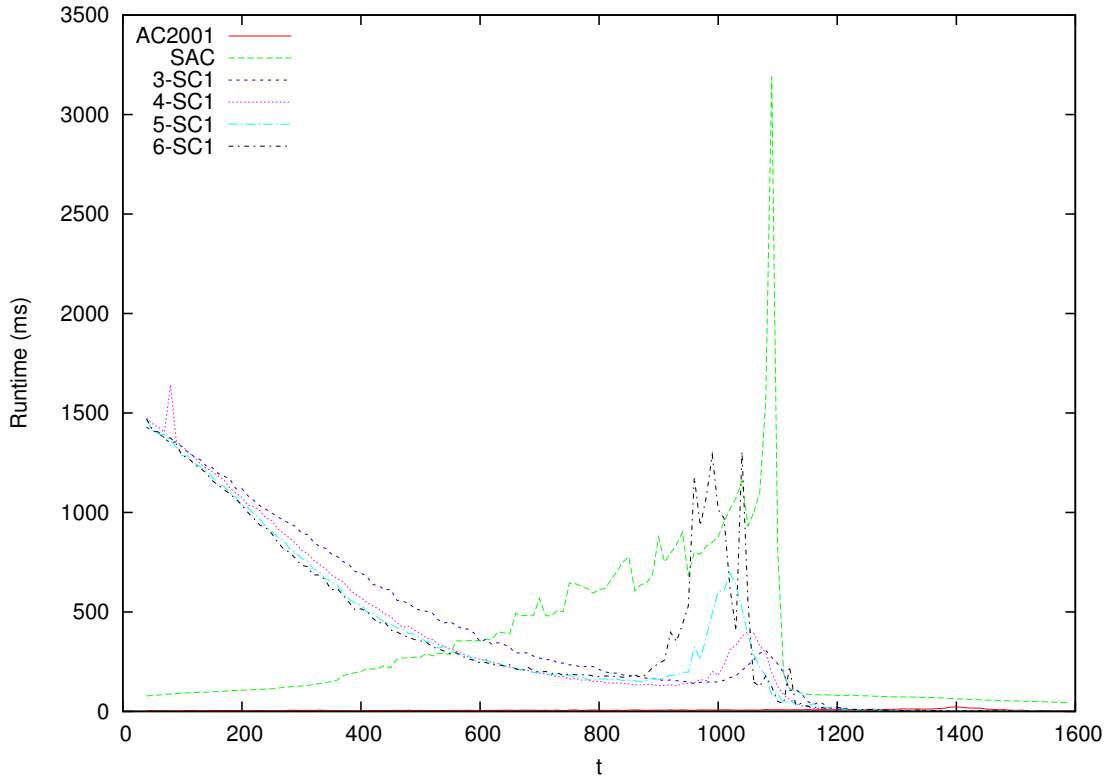


(a)

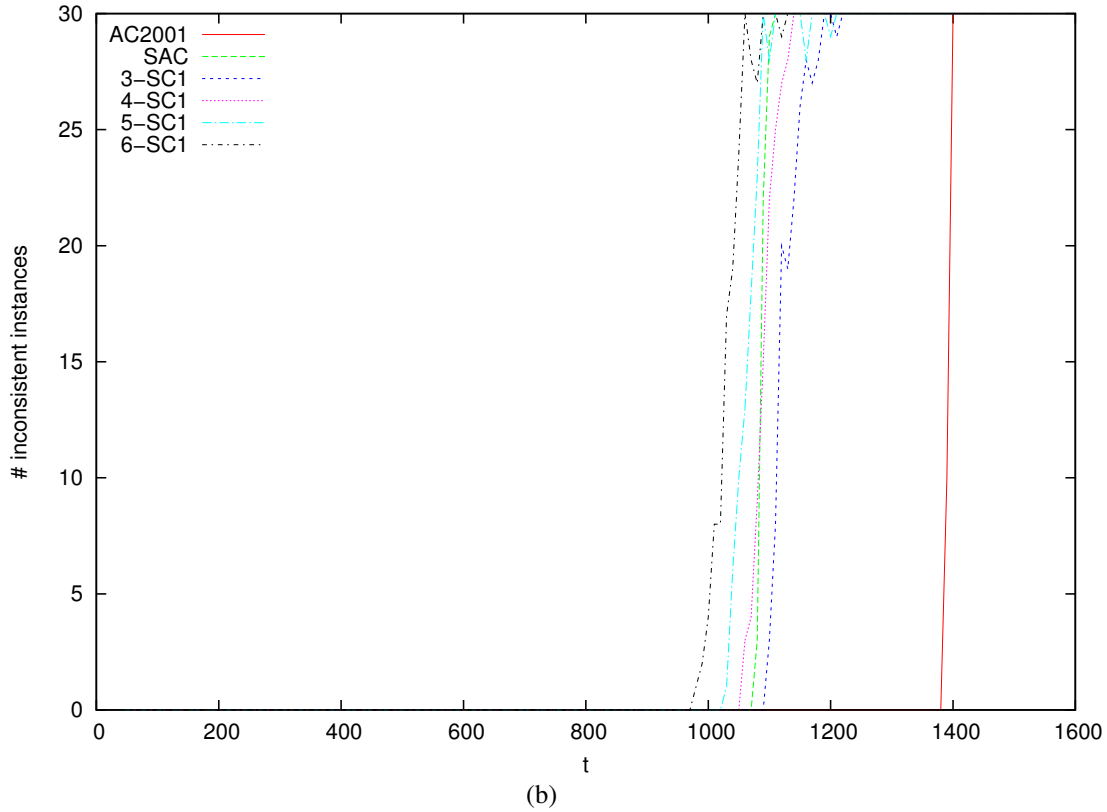


(b)

Figure 3: Results obtained for random instances of classes (200,20,5970,t): (a) runtime in ms and (b) number of detected inconsistent instances.

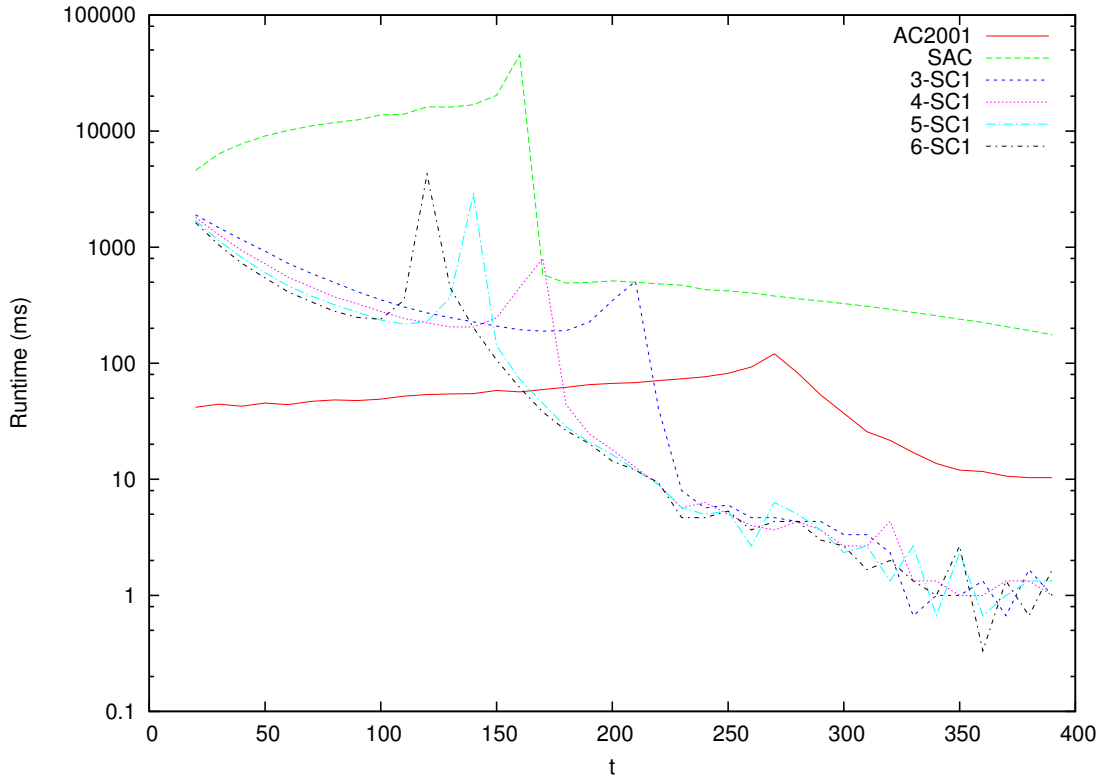


(a)

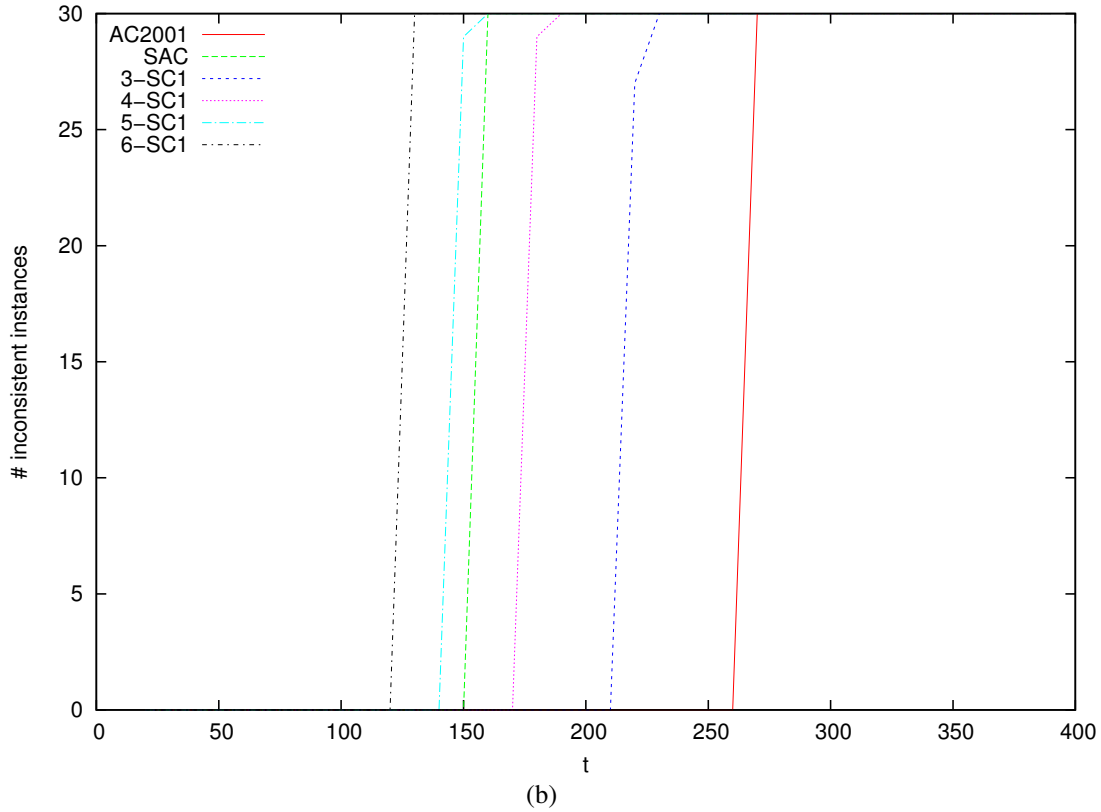


(b)

Figure 4: Results obtained for random instances of classes (100,40,990,t) with 30 instances per class: (a) runtime in ms and (b) number of detected inconsistent instances.



(a)



(b)

Figure 5: Results obtained for random instances of classes (200,20,19900,t) with 30 instances per class: (a) runtime in ms (with a logarithmic scale) and (b) number of detected inconsistent instances.