Exploiting Past and Future: Pruning by Inconsistent Partial State Dominance

Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary, and Vincent Vidal

CRIL – CNRS FRE 2499,
rue de l'université, SP 16
62307 Lens cedex, France
$\{lecoutre, sais, tabary, vidal\}$@cril.univ-artois.fr

Abstract. It has recently been shown, for the Constraint Satisfaction Problem (CSP), that the state associated with a node of the search tree built by a backtracking algorithm can be exploited, using a transposition table, to prevent the exploration of similar nodes. This technique is commonly used in game search algorithms, heuristic search or planning. Its application is made possible in CSP by computing a partial state – a set of meaningful variables and their associated domains – preserving relevant information. We go further in this paper by providing two new powerful operators dedicated to the extraction of inconsistent partial states. The first one eliminates any variable whose current domain can be deduced from the partial state, and the second one extracts the variables involved in the inconsistency proof of the subtree rooted by the current node. Interestingly, we show these two operators can be safely combined, and that the pruning capabilities of the recorded partial states can be improved by a dominance detection approach (using lazy data structures).

1 Introduction

Backtracking search is considered as one of the most successful paradigm for solving instances of the Constraint Satisfaction Problem (CSP). Many improvements have been proposed over the years. They mainly concern three research areas: search heuristics, inference and conflict analysis. To prevent future conflicts, many approaches have been proposed such as intelligent backtracking (e.g. Conflict Based Backjumping [19]), and nogood recording or learning [11].

In the context of satisfiability testing (SAT), studies about conflict analysis have given rise to a new generation of efficient conflict driven solvers called CDCL (Conflict Driven Clause Learning), e.g. Zchaff [22] and Minisat [8]. This has led to a breakthrough in the practical solving of real world instances. The progress obtained within the SAT framework has certainly contributed to a renewed interest of the CSP community to learning [15, 4, 16].

In [17], a promising approach, called state-based search, related to learning, has been proposed. The concept of transposition table, widely used in game search algorithms and planning, has been adapted to constraint satisfaction.

More precisely, at each node of the search tree proved to be inconsistent, a partial snapshot of the current state (a set of meaningful variables and their associated domains) is recorded, in order to avoid a similar situation to occur later during search. To make this approach quite successful, two important issues must be addressed: limiting the space memory required, and improving the pruning capabilities of recorded partial states. Extracting partial states as small as possible (in terms of variables) is the answer to both issues. Different state extraction operators, which have yielded promising practical results, have been proposed in [17].

In this paper, we go further by providing two new powerful extraction operators. The first one eliminates any variable whose current domain can be deduced from the partial state, and the second one extracts the variables involved in the inconsistency proof of the subtree rooted by the current node. Interestingly, we show that these two operators can be combined. Also, we improve the pruning capabilities of the recorded inconsistent partial states using an advanced data structure based on the SAT well-known watched literal technique, and dominance detection (only equivalence detection was addressed in [17]).

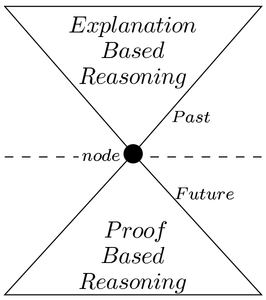


Fig. 1. Partial state extraction from past and future.

Figure 1 illustrates our approach. At each node of the search tree, past and future can be exploited. On the one hand, by collecting information from the past, i.e. on the path leading from the root to the current node, redundant variables of any identified partial state (extracted using any other operator) can be removed. This is called explanation-based reasoning. On the other hand, by collecting information from the future, i.e during the exploration of the subtree, variables not involved in the proof of unsatisfiability can be removed. This is called proof-based reasoning. Pruning by inconsistent state dominance is made more efficient through the combination of both reasonings.

2 Technical Background

A Constraint Network (CN) P is a pair $(\mathscr{X},\mathscr{C})$ where $\mathscr{X}$ is a finite set of n variables and $\mathscr{C}$ a finite set of e constraints. Each variable $X \in \mathscr{X}$ has an associated domain, denoted $dom^P(X)$ or simply $dom(X)$, which contains the set of values allowed for X. The set of variables of P will be denoted by $vars(P)$. A pair (X, a) with $X \in \mathscr{X}$ and $a \in dom(X)$ will be called a value of P. An instantiation t of a set $\{X_1, ..., X_q\}$ of variables is a set $\{(X_i, v_i) \mid i \in [1, q]$ and $v_i \in dom(X_i)\}$. The value v_i assigned to X_i in t will be denoted by $t[X_i]$. Each constraint $C \in \mathscr{C}$ involves a subset of variables of $\mathscr{X}$, called scope and denoted $scp(C)$, and has an associated relation, denoted $rel(C)$, which represents the set of instantiations allowed for the variables of its scope. A constraint $C \in \mathscr{C}$ with $scp(C) = \{X_1, \ldots, X_r\}$ is universal in P iff $\forall v_1 \in dom(X_1), \ldots, \forall v_r \in dom(X_r), \exists t \in rel(C)$ such that $t[X_1] = v_1, \ldots, t[X_r] = v_r$. If P and Q are two

CNs defined on the same sets of variables $\mathscr{X}$ and constraints $\mathscr{C}$, we will write $P \preceq Q$ iff $\forall X \in \mathscr{X}$, $dom^P(X) \subseteq dom^Q(X)$. A solution to P is an instantiation of $vars(P)$ such that all the constraints are satisfied. The set of all the solutions of P is denoted $sol(P)$, and P is satisfiable iff $sol(P) \neq \emptyset$.

The Constraint Satisfaction Problem (CSP) is the NP-complete task of determining whether or not a given CN is satisfiable. A CSP instance is then defined by a CN, and solving it involves either finding one (or more) solution or determining its unsatisfiability. To solve a CSP instance, one can modify the CN by using inference or search methods. Usually, domains of variables are reduced by removing inconsistent values, i.e. values that cannot occur in any solution. Indeed, it is possible to filter domains by considering some properties of constraint networks. Generalized Arc Consistency (GAC) remains the central one. It is for example maintained during search by the algorithm MGAC, called MAC in the binary case.

From now on, we consider a backtracking search algorithm (e.g. MGAC) using a binary branching scheme, in order to solve a given CN P. This algorithm builds a tree search and employs an inference operator ϕ that enforces a domain filtering consistency [6] at any step of the search called ϕ-search. We assume that ϕ satisfies some usual properties such as monotony and confluence. $\phi(P)$ is the CN derived from P obtained after applying the inference operator ϕ. If there exists a variable with an empty domain in $\phi(P)$ then P is clearly unsatisfiable, denoted $\phi(P) = \bot$. Given a set of decisions Δ, $P|_\Delta$ is the CN derived from P such that, for any positive decision $(X = v) \in \Delta$, $dom(X)$ is restricted to $\{v\}$, and, for any negative decision $(X \neq v) \in \Delta$, v is removed from $dom(X)$.

We assume that any inference is performed locally, i.e. at the level of a single constraint C, during the propagation achieved by ϕ. The inference operator ϕ can be seen as a collection of local propagators associated with each constraint, called ϕ-propagators. These propagators can correspond to either the generic revision procedure of a coarse-grained GAC algorithm called for a constraint, or to a specialized filtering procedure (e.g. in the context of global constraints).

3 Inconsistent Partial States

In this section, we introduce the central concept of partial state of a constraint network P. It corresponds to a set of variables of P with their potentially reduced associated domains.

Definition 1. *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN, a partial state Σ of P is a set of pairs (X, D_X) with $X \in \mathscr{X}$ and $D_X \subseteq dom^P(X)$ such that any variable of $\mathscr{X}$ appears at most once in Σ.*

The set of variables occurring in a partial state Σ is denoted by $vars(\Sigma)$, and for any $(X, D_X) \in \Sigma$, $dom^\Sigma(X)$ denotes D_X. At a given step of a backtracking search, a partial state can be associated with the corresponding node of the search tree. This partial state is naturally built by taking into account all variables and their current domains, and will be called the current state.

A network can be restricted over one of its partial state Σ by replacing in P the domain of each variable occurring in Σ with its corresponding domain in Σ. The restricted network is clearly smaller ($\preceq$) than the initial network.

Definition 2. *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN and Σ be a partial state of P. The restriction $\psi(P, \Sigma)$ of P over Σ is the CN $P' = (\mathscr{X}, \mathscr{C})$ such that $\forall X \in \mathscr{X}$, $dom^{P'}(X) = dom^{\Sigma}(X)$ if $X \in vars(\Sigma)$, and $dom^{P'}(X) = dom^{P}(X)$ otherwise.*

A partial state Σ of a CN P is said to be inconsistent with respect to P when the network defined as the restriction of P over Σ is unsatisfiable.

Definition 3. *Let P be a CN and Σ be a partial state of P. Σ is an inconsistent partial state of P (IPS_P for short), iff $\psi(P, \Sigma)$ is unsatisfiable.*

A partial state Σ is said to dominate a CN P if each variable of Σ occurs in P with a smaller domain.

Definition 4. *Let P and P' be two CNs such that $P' \preceq P$, and Σ be a partial state of P. Σ dominates P' iff $\forall X \in vars(\Sigma)$, $dom^{P'}(X) \subseteq dom^{\Sigma}(X)$.*

The following proposition is at the core of state-based reasoning by dominance detection.

Proposition 1. *Let P and P' be two CNs such that $P' \preceq P$, and Σ be an IPS_P. If Σ dominates P', P' is unsatisfiable.*

Proof. It is immediate since we can easily deduce that $\psi(P', \Sigma) \preceq \psi(P, \Sigma)$ from $P' \preceq P$ and the definition of ψ. $\square$

In the context of solving a constraint network P using a backtracking search, this proposition can be exploited to prune nodes dominated by previously identified IPS_P. An IPS_P can be extracted from a node proved to be the root of an unsatisfiable subtree. Although the current state associated to such a node is an IPS_P itself, it cannot obviously be encountered later during search: to be useful, it must be reduced. That is why in what follows, we propose two new operators (and adapt an existing one) which eliminate some variables from a state proved unsatisfiable, while preserving its status of IPS_P.

Finally, it is important to relate the concept of (inconsistent) partial state with those of Global Cut Seed [10] and pattern [9]. The main difference is that a partial state can be defined from a subset of variables of the CN, whereas GCS and patterns, introduced to break global symmetries, contain all variables of the CN. On the other hand, it is close to the notion of nogood as introduced in [20], where it is shown that a nogood can be reduced to a subset of variables, those involved in the decisions taken along the current branch leading to an inconsistent node. In our context, we will show that it is possible to build a partial state by removing variables involved or not in taken decisions.

4 Universality-based Extraction

In [17], three different operators have been introduced to extract *constraint subnetworks* whose state can be recorded in a transposition table. These operators allow to remove so-called s-eliminable (ρ^{sol}), u-eliminable (ρ^{uni}) and r-eliminable (ρ^{red}) variables. The principle is to determine whether or not the subnetwork corresponding to the current node of the search tree is equivalent to one already recorded in the table. If this is the case, this node can be safely discarded.

In this paper, we apply this approach to state dominance detection and propose new advanced operators. Only, the ρ^{uni} operator will be considered in our context, as s-eliminable variables are also u-eliminable (ρ^{sol} is interesting to count solutions), and the removal of r-eliminable variables is immediate when considering dominance detection. We propose a new formulation of this operator.

Definition 5. *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN. The operator $\rho^{uni}(P)$ denotes the partial state $\Sigma = \{(X, dom^P(X)) \mid X \in \mathscr{X}$ and $\exists C \in \mathscr{C} \mid X \in scp(C)$ and C is not universal in $P\}$. A variable $X \in \mathscr{X} \setminus vars(\Sigma)$ is called an u-eliminable variable of P.*

The following proposition establishes that ρ^{uni} can extract inconsistent partial states at any node of a search tree proved to be the root of an unsatisfiable subtree.

Proposition 2. *Let P and P' be two CNs such that $P' \preceq P$. If P' is unsatisfiable then $\rho^{uni}(P')$ is an IPS$_P$.*

Proof. As shown in [17], the constraint subnetwork defined from the variables of $\Sigma = \rho^{uni}(P')$ is unsatisfiable. Its embedding in any larger constraint network entails its unsatisfiability. □

5 Proof-based Extraction

Not all constraints of an unsatisfiable constraint network are necessary to prove its unsatisfiability. Some of them form (Minimal) Unsatisfiable Cores (MUCs) and different methods have been proposed to extract them. Indeed, one can iteratively identify the constraints of a MUC following a constructive [5], a destructive [1] or a dichotomic approach [14, 13]. More generally, an unsatisfiable core can be defined as follows:

Definition 6. *Let $P = (\mathscr{X}, \mathscr{C})$, $P' = (\mathscr{X}', \mathscr{C}')$ be two CNs. P' is an unsatisfiable core of P if P' is unsatisfiable, $\mathscr{X}' \subseteq \mathscr{X}$, $\mathscr{C}' \subseteq \mathscr{C}$ and $\forall X' \in \mathscr{X}', dom^{P'}(X') = dom^P(X')$.*

Interestingly, it is possible to associate an IPS$_P$ with any unsatisfiable core extracted from a network $P' \preceq P$. This is stated by the following proposition.

Proposition 3. *Let P and P' be two CNs such that $P' \preceq P$. For any unsatisfiable core P'' of P', $\Sigma = \{(X, dom^{P''}(X)) \mid X \in vars(P'')\}$ is an IPS$_P$.*

Proof. If Σ is not an IPS$_P$, i.e. if $\psi(P, \Sigma)$ is satisfiable, there exists an assignment of a value to all variables of $vars(\Sigma)$ such that all constraints of P are satisfied. As any constraint of P'' is included in P', and so in P, this contradicts our hypothesis of P'' being an unsatisfiable core. □

As an inconsistent partial state can be directly derived from an unsatisfiable core, one can be interested in extracting such cores from any node proved to be the root of an unsatisfiable subtree. Computing a posteriori a MUC from scratch using one of the approaches mentioned above seems very expensive since even the dichotomic approach is in $O(log(e).k_e)$ [13] where k_e is the number of constraints of the extracted core. However, it is possible to efficiently identify an unsatisfiable core by keeping track of all constraints involved in the proof of unsatisfiability [1]. Such constraints are the ones used during search to remove, through their propagators, at least one value in the domain of one variable. We adapt this "proof-based" approach to extract an unsatisfiable core from any node of the search tree by incrementally collecting relevant information.

Algorithm 1 depicts how to implement our method inside a backtracking ϕ-search algorithm. The recursive function *solve* determines the satisfiability of a network P and returns a pair composed of a Boolean (that indicates if P is satisfiable or not), and a set of variables. This set is either empty (when P is satisfiable) or represents a proof of unsatisfiability. A proof is composed of the variables involved in the scope of the constraints that triggered at least one removal during ϕ-propagation.

At each node, a proof, initially empty, is built from all inferences produced when enforcing ϕ and the proofs (lines 6 and 8) associated with the left and right subtrees (once a pair (X, a) has been selected). When the unsatisfiability of a node is proved after having considered two branches (one labelled with $X = a$ and the other with $X \neq a$), one obtains a proof of unsatisfiability (line 10) by simply merging the proofs associated with the left and right branches. Remark that the worst-case space complexity of managing the different local proofs of the search tree is in $O(n^2 d)$ since storing a proof is $O(n)$ and there are at most $O(nd)$ nodes per branch.

Algorithm 1: $solve(P = (\mathscr{X}, \mathscr{C})$: CN): (Boolean, Set of Variables)

1 $localProof \leftarrow \emptyset$
2 $P' = \phi(P)$ // localProof updated according to ϕ
3 **if** $P' = \bot$ **then return** $(false, localProof)$
4 **if** $\forall X \in \mathscr{X}, |dom(X)| = 1$ **then return** $(true, \emptyset)$
5 select a pair (X, a) with $|dom(X)| > 1 \wedge a \in dom(X)$
6 $(sat, leftProof) \leftarrow solve(P'|_{X=a})$
7 **if** sat **then return** $(true, \emptyset)$
8 $(sat, rightProof) \leftarrow solve(P'|_{X \neq a})$
9 **if** sat **then return** $(true, \emptyset)$
10 // $leftProof \cup rightProof$ is a proof of inconsistency for P'
11 **return** $(false, localProof \cup leftProof \cup rightProof)$

Using Algorithm 1, we can introduce a new advanced extraction operator that only retains variables involved in a proof of unsatisfiability. This operator can be incrementally used at any node of a search tree proved to be the root of an unsatisfiable subtree.

Definition 7. *Let P be a CN such that $solve(P) = (false, proof)$. The operator $\rho^{prf}(P)$ denotes the partial state $\Sigma = \{(X, dom^P(X)) \mid X \in proof\}$. A variable $X \in vars(P) \setminus vars(\Sigma)$ is called a p-eliminable variable of P.*

Proposition 4. *Let P and P' be two CNs such that $P' \preceq P$. If P' is unsatisfiable then $\rho^{prf}(P')$ is an IPS$_P$.*

Proof. Let $P = (\mathscr{X}, \mathscr{C})$ and $solve(P') = (false, proof)$. Clearly, $P'' = (proof, \{C \in \mathscr{C} \mid scp(C) \subseteq proof\})$ is an unsatisfiable core of P'. $\rho^{prf}(P')$ is equal to $\{(X, dom^{P'}(X)) \mid X \in proof\}$ which is proved to be an IPS$_P$ by Prop. 3. □

In practice, in Algorithm 1, one can call the ρ^{prf} operator to extract an IPS$_P$ at line 10. Interestingly enough, the following proposition establishes that ρ^{prf} is stronger than ρ^{uni} (i.e. allows to extract partial states representing larger portions of the search space).

Proposition 5. *Let P be an unsatisfiable CN. $\rho^{prf}(P) \subseteq \rho^{uni}(P)$.*

Proof. An universal constraint cannot occur in an unsatisfiability proof. An u-eliminable variable only occurs in universal constraints, so is p-eliminable. □

It must be noted that, unlike ρ^{uni}, extracting an inconsistent partial state using ρ^{prf} can only be performed when the subtree has been completely explored. As a consequence, it is not possible to use this operator for pruning equivalent states using a transposition table whose keys correspond to partial states. Nevertheless, ρ^{prf} can be fully exploited in the context of dominance detection.

6 Explanation-based Extraction

In this section, we propose a second advanced extraction operator of (inconsistent) partial states. Unlike the proof-based extraction operator, this new one can be applied each time we reach a new node by analyzing all propagation performed so far. The principle of this operator is to build a partial state by eliminating the variables whose domains can be inferred from the other ones. This is made possible by keeping track, for any value removed from the initial network, of the constraint at the origin of its removal. This kind of original explanations can be related to the concept of implication graphs used in the SAT community. In the context of achieving arc consistency for dynamic CSPs [2], such explanations were also exploited to put values back into domains when constraints are retracted.

In constraint satisfaction, eliminating explanations are classically decision-based, which means that each removed value is explained by a set of positive decisions, i.e. a set of variable assignments. This is usually exploited to perform some kind of intelligent backtracking (e.g. [7, 19, 12]). Interestingly, it is possible to define explanations in a more general way by taking into account not only some decisions taken during search but also some constraints of the original network.

Explanations recorded for each removal can be represented using an implication graph as used in the satisfiability context [22]. Given a set of decisions (the current partial instantiation), the inference process can be modelled using a fine-grained implication graph. More precisely, for each removed value, one can record positive and negative decisions implying this removal (through clauses). For our purpose, we can simply reason using a coarse-grained level of the implication graph. Whenever a value (X, a) is removed during the propagation associated with a constraint C, the (local) eliminating explanation of (X, a) is simply given by C. In other words, as our aim is to circumscribe a partial state (smaller than the current state in terms of variables), we only need to know for each removed value (X, a), the variables (in fact, those involved in C) responsible of its removal. From this information, it is possible to build a directed graph G where nodes correspond to variables and arcs to dependencies between variables. More precisely, an arc exists in G from a variable Y to a variable X if there exists a removed value (X, a) such that its local explanation is a constraint involving Y. Also, a special node denoted by nil is considered, and an arc exists from nil to a variable X if this variable is involved in a (positive or negative) decision. The implication graph can then be used to extract an inconsistent partial state from a subset of variables S (that already represents an inconsistent partial state) by eliminating any variable with no incoming arc from a variable outside S. Of course, such an extraction is not interesting if S is $\mathscr{X}$ since it necessary produces a set of variables corresponding to all decisions.

Important: For all definitions and propositions below, we consider given two CNs $P = (\mathscr{X}, \mathscr{C})$ and P' such that P' corresponds to a node of the ϕ-search tree of P. We obviously have $P' \preceq P$.

Definition 8. *For any (X, a) such that $X \in \mathscr{X}$ and $a \in dom^P(X) \setminus dom^{P'}(X)$, the local eliminating explanation of (X, a), denoted by $exp(X, a)$ is, if it exists, the constraint C whose associated ϕ-propagator has removed (X, a) along the path leading from P to P', and nil otherwise.*

These explanations can be used to extract a partial state from a CN wrt P and a set of variables S. This partial state contains the variables of S that cannot be "explained" by S.

Definition 9. *$\forall S \subseteq \mathscr{X}$, $\rho^{exp}_{P,S}(P')$ is the partial state $\Sigma = \{(X, dom^{P'}(X)) \mid X \in S$ and $\exists a \in dom^P(X) \setminus dom^{P'}(X)$ such that $(exp(X, a) = nil$ or $\exists Y \in scp(exp(X, a))$ such that $Y \notin S)\}$. A variable $X \in S \setminus vars(\Sigma)$ is called an i-eliminable variable of P' wrt P and S.*

Proposition 6. *Let Σ be a partial state of P' and $\Sigma' = \rho^{exp}_{P,vars(\Sigma)}(P')$. We have: $\phi(\psi(P, \Sigma')) = \phi(\psi(P, \Sigma))$.*

Proof sketch. The only variables whose domain may differ between $\psi(P,\Sigma')$ and $\psi(P,\Sigma)$ are the i-eliminable variables of the set $\Delta = vars(\Sigma) \setminus vars(\Sigma')$. We also have $\forall X \in vars(\Sigma'), dom^{\psi(P,\Sigma')}(X) = dom^{\psi(P,\Sigma)}(X) = dom^{P'}(X)$ and $\forall X \in \Delta, dom^{\psi(P,\Sigma')}(X) = dom^{P}(X)$. This means that the domains of the variables in Σ' are in the state they were after performing all decisions and propagations that lead to P', and the domains of the variables of Δ are reset to the state they were in P. Also, every variable $X \in vars(P) \setminus vars(\Sigma)$ is such that $dom^{\psi(P,\Sigma')}(X) = dom^{\psi(P,\Sigma)}(X)$.

Let $R_\Delta = \{(X,a) \mid X \in \Delta \wedge a \in dom^P(X) \setminus dom^{P'}(X)\}$ be the set of values removed for the variables of Δ on the branch leading from P to P'. For any $(X,a) \in R_\Delta$, we have an explanation $C = exp(X,a)$ such that $C \neq nil$ and $scp(C) \subseteq vars(\Sigma)$ since X is i-eliminable. In other words, the removal of values from R_Δ were triggered along the path leading to P' by constraints (the explanations) only involving variables of Σ, that is variables of Σ' and Δ itself.

In $\psi(P,\Sigma')$, we can trigger the removal of all values in R_Δ in the order they were performed along the path leading to P'. Indeed, following the same order, the explanation associated with each value of R_Δ can trigger its removal again as (1) the domains of the variables of Σ' are kept after their reduction in the branch leading to P' ($\forall X \in \Sigma', dom^{\psi(P,\Sigma')}(X) = dom^{P'}(X)$), (2) variables of Δ are reset to their state in P ($\forall X \in \Delta, dom^{\psi(P,\Sigma')}(X) = dom^{P}(X)$) and (3) the explanation of any removal only involves variables of Σ. This can be shown by a recurrence on the order values are removed in the branch leading to P'. Finally, as the removed values represent the only difference between $\psi(P,\Sigma')$ and $\psi(P,\Sigma)$, by confluence of ϕ, we can conclude that $\phi(\psi(P,\Sigma')) = \phi(\psi(P,\Sigma))$. □

The following corollary (whose proof is a direct consequence of Proposition 6) is particularly interesting since it states that we can safely use ρ^{exp} after any other one which produces an IPS$_P$.

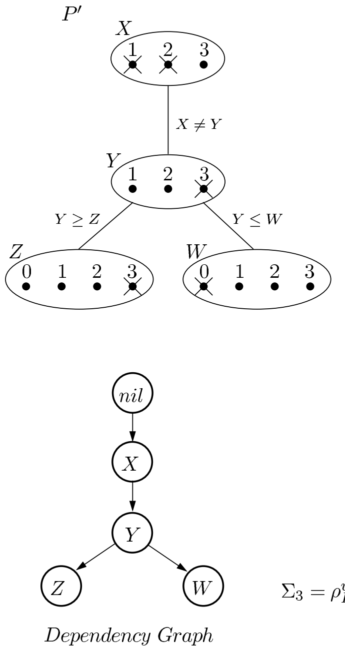
Corollary 1. *Let Σ be a partial state of P' and $\Sigma' = \rho^{exp}_{P,vars(\Sigma)}(P')$. If Σ is an IPS$_P$ then Σ' is an IPS$_P$.*

It follows that the next two operators are guaranteed to produce an IPS$_P$.

Definition 10. $\rho^{prex}_P(P') = \rho^{exp}_{P,vars(\Sigma)}(P')$ *with* $\Sigma = \rho^{prf}(P')$. $\rho^{unex}_P(P') = \rho^{exp}_{P,vars(\Sigma)}(P')$ *with* $\Sigma = \rho^{uni}(P')$.

The ρ^{exp} operator can be implemented with a two-dimensional array exp such that for any pair (X,a) removed from P during a ϕ-search, $exp[X,a]$ represents its local eliminating explanation. When a positive decision $X = a$ is taken, $exp[X,b] \leftarrow nil$ for all remaining values $b \in dom(X) \mid b \neq a$, and when a negative decision $X \neq a$ is taken, $exp[X,a] \leftarrow nil$. The space complexity of exp is $O(nd)$ while the time complexity of managing this structure is $O(1)$ whenever a value is removed or restored during search. The worst-case time complexity of ρ^{exp} is $O(ndr)$ where r denotes the greatest constraint arity. Indeed, there are at most $O(nd)$ removed values which admit a local eliminating explanation.

Figure 1 illustrates (on consistent partial states) the behavior of ρ^{uni}, ρ^{exp} and their combination ρ^{unex}. The problem at hand involves four variables (X,



$$exp(X,1) = nil$$
$$exp(X,2) = nil$$
$$exp(Y,3) = (X \neq Y)$$
$$exp(Z,3) = (Y \geq Z)$$
$$exp(W,0) = (Y \leq W)$$

$$P \rightarrow \left\{ \begin{array}{l} X,\{1,2,3\} \\ Y,\{1,2,3\} \\ Z,\{0,1,2,3\} \\ W,\{0,1,2,3\} \end{array} \right\}$$

$$P' = \phi(P|_{X=3}) \rightarrow \left\{ \begin{array}{l} X,\{3\} \\ Y,\{1,2\} \\ Z,\{0,1,2\} \\ W,\{1,2,3\} \end{array} \right\}$$

$$\Sigma_1 = \rho^{uni}(P') \rightarrow \left\{ \begin{array}{l} Y,\{1,2\} \\ Z,\{0,1,2\} \\ W,\{1,2,3\} \end{array} \right\}$$

$$\Sigma_2 = \rho^{exp}_{P,vars(P)}(P') \rightarrow \{\ X,\{3\}\ \}$$

$$\Sigma_3 = \rho^{unex}_{P}(P') = \rho^{exp}_{P,vars(\Sigma_1)}(P') \rightarrow \{\ Y,\{1,2\}\ \}$$

Fig. 2. Extracting partial states using ρ^{uni}, ρ^{exp} and ρ^{unex}.

Y, Z, W) and three constraints ($X \neq Y$, $Y \geq Z$, $Y \leq W$). When the decision $X = 3$ is taken, the explanation associated with the removal of 1 and 2 from $dom(X)$ is set to nil. These removals are propagated to Y through the constraint $X \neq Y$, yielding the removal of 3 from $dom(Y)$. The explanation of this removal is thus set to $X \neq Y$. This removal is then propagated to Z and W: 3 is removed from $dom(Z)$ through the propagation of $Y \geq Z$ which constitutes its explanation, and 0 is removed from $dom(W)$ through the propagation of $Y \leq W$. No more propagation is possible, and the resulting network is denoted by P'. The dependency graph exploited later by ρ^{exp} is then built from these explanations.

Applying ρ^{uni} to P' leads to the elimination of X yielding the partial state Σ_1, as X is now only involved in universal constraints. Indeed, the remaining value 3 in $dom(X)$ is compatible with the two remaining values 1 and 2 in $dom(Y)$ within the constraint $X \neq Y$. The three other variables are involved in constraints which are not universal.

Applying ρ^{exp} to P' and $S = vars(P)$ leads to the elimination of Y, Z and W, yielding the partial state $\Sigma_2 = \{(X, \{3\})\}$. Indeed, X is the only variable from which a removal is explained by nil (S being all variables of P', this is the only relevant condition for determining variables of interest). This illustrates the fact that applying ρ^{exp} to all variables of a constraint network has no interest: as we obtain the set of decisions of the current branch, the partial state can never be encountered (or dominated) again without restarts. Note that we would have obtained the same result with a classical decision-based explanation scheme.

More interesting is the application of ρ^{unex}. Once ρ^{uni} has been applied, giving the partial state Σ_1 whose variables are $\{Y, Z, W\}$, ρ^{exp} is applied to determine which variables of Σ_1 have domains that can be determined by other variables of Σ_1. The variable Y is the only one for which all removals cannot be explained by constraints whose scope involve variables *inside* Σ_1 only, as the explanation $X \neq Y$ of the removal $(Y, 3)$ involves now a variable *outside* the variables of interest. This yields the partial state $\Sigma_3 = \{(Y, \{1, 2\})\}$, that contains a variable which is not a decision, and which can then be exploited later during search.

7 Dominance State Detection

In the context of a ϕ-search algorithm, we give now some details about the exploitation of the reduction operators. At each node associated with an unsatisfiable network, one can apply one (or a combination) of the operators to extract an inconsistent partial state, and record it in an IPS_P base. The IPS_P can then be exploited later during search either to prune nodes of the search tree, or to make additional inferences.

Equivalent nodes can be pruned using transposition tables as proposed in [17], but ρ^{prf} cannot be exploited this way. Indeed, when a node is opened, computing a key for it (to be used in the transposition table) is impossible: it requires the complete exploration of the subtree rooted by this node. As such, equivalence detection through the transposition table cannot be performed. However, a node dominated by an IPS_P stored in the base can be safely pruned.

One can go further, by identifying the values whose presence would lead to expand nodes dominated by an IPS_P. Similarly to [16], such inferences can be done thanks to the lazy data structure of watched literals [18] used to manage the IPS_P base. A watched literal of an IPS_P Σ is a pair (X, a) such that $X \in vars(\Sigma)$ and $a \in dom^{\Sigma}(X)$. It is said to be valid when $a \in dom(X) \setminus dom^{\Sigma}(X)$. Two watched literals are associated with each inconsistent partial state Σ. Σ is valid if its two watched literals are valid. When a value a is removed from $dom(X)$, the set of the IPS_P where (X, a) is watched is not valid anymore. To maintain the validity of these inconsistent partial states, we must for each of them, either find another valid watched literal, or remove the values in $dom(Y) \cap dom^{\Sigma}(Y)$ where (Y, b) is the second watched literal. Exploiting this structure, we have the guarantee that the current node cannot be dominated by an IPS_P.

Note that, when using the ρ^{prf} operator, inferences must be performed with caution. Indeed, the IPS_P Σ responsible of an inference participates to the proof of unsatisfiability of the current node. Σ can be seen as an additional constraint of the initial network: each variable occurring in $vars(\Sigma)$ must then also occur in the proof. Finally, whatever the operators are used, variables whose current domain has never been reduced on the current branch can be safely eliminated. Indeed, the dominance for such variables is guaranteed to hold.

Table 1. Number of solved instances per series (1, 800 seconds allowed per instance).

Series	*#Inst*	*brelaz*			*dom/ddeg*			*dom/wdeg*		
		$\neg\rho$	ρ^{uni}	ρ^{prex}	$\neg\rho$	ρ^{uni}	ρ^{prex}	$\neg\rho$	ρ^{uni}	ρ^{prex}
aim	48	32	25 (29)	39	32	25 (29)	38	48	43 (47)	48
dubois	13	4	0 (2)	13	4	1 (2)	13	5	13 (3)	11
ii	41	10	10 (10)	13	10	9 (10)	16	20	18 (19)	31
os-taillard-10	30	5	5 (5)	5	4	4 (4)	4	10	10 (10)	13
pigeons	25	13	17 (19)	13	13	17 (19)	13	13	16 (18)	10
pret	8	4	4 (4)	8	4	4 (4)	8	4	8 (4)	8
ramsey	16	3	3 (3)	6	5	3 (5)	5	6	5 (6)	6
scens-11	12	0	0 (0)	0	0	0 (0)	4	9	7 (8)	9
	193	71	64 (72)	92	73	63 (73)	105	115	120 (115)	136

8 Experiments

In order to show the practical interest of the new operators introduced for dominance detection, we have conducted an experimentation on a Xeon processor cadenced at 3 GHz and 1GiB RAM. We have used benchmarks from the second CSP solver competition (http://cpai.ucc.ie/06/Competition.html) including binary and non binary constraints expressed in extensional and intentional form. We have used MGAC (in our solver Abscon[1]) with various combinations of extraction operators and variable ordering heuristics. Performance is measured in terms of cpu time in seconds (*cpu*), number of visited nodes (*nodes*), memory in MiB (*mem*) and average number of variables eliminated when building inconsistent partial states (*elim*). For ρ^{uni}, we considered the same restriction as the one mentioned in [17]: only the variables with a singleton domain involved in constraints binding at most one non singleton-domain variable are removed (to avoid checking the universality of constraints). We also experimented equivalence detection (using a transposition table) with the operator ρ^{red} proposed in [17]: as ρ^{red} is related to ρ^{uni} since they have the same behaviour for dominance detection, the obtained results are given between brackets in the columns of ρ^{uni}.

Table 1 presents the results obtained on some series of structured instances. We do not provide any results on random instances as, unsurprisingly, our learning approach is not adapted to them. The tested configurations are labelled $\neg\rho$ (MGAC without state-based reasoning), ρ^{uni} and ρ^{prex}, each one being combined with the three heuristics *brelaz*, *dom/ddeg* and *dom/wdeg* [3]. The first thing that we can observe is that, whatever the heuristic is used, more instances are solved using ρ^{prex}. Also, note that the performance of the dominance detection approach can be damaged when ρ^{uni} is used: more instances are solved with *brelaz* and *dom/ddeg* using equivalence detection (results between brackets). Indeed, for ρ^{uni}, the size of the IPS_P can often be quite high, which directly affects dominance checking; whereas equivalence detection can be performed in nearly constant time using a hash table.

Table 2 focuses on some instances with the same tested configurations (*brelaz* is omitted, as similar results are obtained with *dom/ddeg*). One can first observe a drastic reduction in the number of expanded nodes using dominance detection,

[1] http://www.cril.univ-artois.fr/~lecoutre/research/tools/abscon.html

Table 2. Results on some structured instances (1,800 seconds allowed per instance).

		dom/ddeg			*dom/wdeg*		
		$\neg\rho$	ρ^{uni}	ρ^{prex}	$\neg\rho$	ρ^{uni}	ρ^{prex}
BlackHole-4-4-e-0	*cpu*	2.39	2.31 (1.9)	2.36	2.3	3.07 (2.25)	3.37
	nodes	6,141	1,241 (1,310)	931	6,293	3,698 (4,655)	5,435
($\#V = 64$)	*elims*	0	13.61 (14.16)	58.88	0	14.35 (14.84)	58.02
aim-100-1-6-1	*cpu*	11.54	65.16 (19.2)	2.45	2.14	2.45 (2.34)	2.2
	nodes	302K	302K (302K)	737	987	998 (909)	616
($\#V = 200$)	*elims*	0	161.80 (161.78)	190.22	0	174.94 (176.18)	189.78
aim-200-1-6-1	*cpu*	–	– (–)	4.059	3.4	4.44 (6.56)	3.18
	nodes			6,558	9,063	7,637 (26,192)	1,814
($\#V = 400$)	*elims*			382.73	0	356.39 (348.35)	388.63
composed-25-10-20-9	*cpu*	7.39	6.02 (6.95)	2.65	2.56	2.77 (2.56)	2.47
	nodes	75,589	20,248 (54,245)	184	323	315 (323)	164
($\#V = 105$)	*elims*	0	48.90 (48.95)	87.75	0	67.02 (65.91)	89.25
driverlogw-09-sat	*cpu*	422.2	189.71 (378.87)	83.85	13.69	14.61 (13.13)	12.25
	nodes	118K	37,258 (98,320)	17,623	12,862	8,592 (11,194)	6,853
($\#V = 650$)	*elims*	0	511.69 (514.14)	541.81	0	513.21 (523.26)	544.05
dubois-20	*cpu*	183.38	110.71 (78.94)	1.4	65.95	1.62 (47.72)	1.96
	nodes	24M	787K (2,097K)	379	8,074K	1,252 (1,660K)	2,133
($\#V = 60$)	*elims*	0	42.00 (48.50)	56.63	0	52.25 (45.93)	51.17
dubois-30	*cpu*	–	– (–)	1.7	–	2.41 (–)	3.39
	nodes			724		4,267	12,190
($\#V = 90$)	*elims*			86		81.2	78.5
ii-8a2	*cpu*	16.87	– (44.23)	4.09	3.08	3.99 (3.69)	2.99
	nodes	214K	(214K)	5,224	4,390	4,391 (4,390)	1,558
($\#V = 360$)	*elims*	0	(276.08)	317.04	0	291.67 (291.90)	316.98
ii-8b2	*cpu*	–	– (–)	7.92	9.16	26.86 (22.48)	6.71
	nodes			2,336	11,148	11,309 (11,148)	3,239
($\#V = 1,152$)	*elims*			1,090	0	979.59 (980.08)	1,050
os-taillard-10-100-3	*cpu*	–	– (–)	–	–	– (–)	467.26
	nodes						134K
($\#V = 100$)	*elims*						66.90
pigeons-15	*cpu*	–	53.34 (5.63)	–	–	882.41 (23.62)	–
	nodes		106K (115K)			517K (900K)	
($\#V = 15$)	*elims*		6.99 (7.49)			8.13 (8.63)	
pret-150-25	*cpu*	–	– (–)	3.11	–	59.66 (6.32)	4.4
	nodes			9,003		203K (97,967)	17,329
($\#V = 150$)	*elims*			135.72		132.62 (133.71)	137.67
pret-60-25	*cpu*	66.71	3.17 (3.38)	1.79	76.57	1.97 (1.94)	2.0
	nodes	7,822K	17,530 (47,890)	1,503	7,752K	2,631 (4,080)	2,501
($\#V = 60$)	*elims*	0	45.56 (45.71)	52.32	0	51.47 (52.46)	51.98
ramsey-16-3	*cpu*	72.72	– (108.41)	18.25	–	– (–)	–
	nodes	1,162K	(1,162K)	46,301			
($\#V = 120$)	*elims*	0	(84.44)	105.49			
ramsey-25-4	*cpu*	3.86	4.18 (4.11)	4.15	3.81	4.17 (4.14)	4.04
	nodes	591	591 (591)	570	590	(591) (590)	537
($\#V = 300$)	*elims*	0	191.62 (191.40)	274.72	0	159.81 (159.73)	269.36
scen11-f6	*cpu*	–	– (–)	371.02	42.15	13.15 (10.17)	5.56
	nodes			110K	217K	16,887 (18,938)	2,585
($\#V = 680$)	*elims*			655.80	0	22.72 (22.17)	654.81

especially with ρ^{prex}. This is mainly due to the high average percentage of eliminated variables from IPS_P (around 90% for ρ^{prex}, much less for ρ^{uni}), which compensates the cost of managing the IPS_P base. The bad results with ρ^{prex} on *pigeons* instances can be explained by the fact that many positive decisions are stored in unsatisfiability proofs when propagating the IPS_P base.

Table 3 exhibits some results obtained for hard RLFAP instances. We only consider *dom/wdeg* here, but with all extraction operators mentioned in this paper. Clearly, the dominance detection approach with ρ^{uni} suffers from mem-

Table 3. Results on hard RLFAP instances using *dom/wdeg* (1,800 seconds allowed).

Instance		$\neg\rho$	ρ^{uni}	ρ^{unex}	ρ^{prf}	ρ^{prex}
	cpu	9.0	10.4 (8.54)	12.3 (9.6)	5.7	5.5
*scen*11-*f*8	*mem*	29	164 (65)	49 (37)	33	33
($\#V = 680$)	*nodes*	15,045	13,319 (13,858)	13,291 (13,309)	1,706	1,198
	elim		39.0 (38.1)	590.2 (590.2)	643.3	656.0
	cpu	26.0	11.1 (9.23)	10.4 (10.06)	5.5	5.6
*scen*11-*f*7	*mem*	29	168 (73)	49 (37)	33	33
($\#V = 680$)	*nodes*	113K	13,016 (14,265)	12,988 (13,220)	2,096	1,765
	elim		25.2 (25.4)	584.1 (584.7)	647.5	654.8
	cpu	41.2	15.0 (10.61)	15.5 (10.14)	6.4	6.8
*scen*11-*f*6	*mem*	29	200 (85)	53 (37)	33	33
($\#V = 680$)	*nodes*	217K	16,887 (18,938)	16,865 (17,257)	2,903	2,585
	elim		22.7 (22.1)	588.6 (589.3)	648.8	654.8
	cpu	202	– (72.73)	195 (98.16)	31.5	12.2
*scen*11-*f*5	*mem*	29	256	342 (152)	53	41
($\#V = 680$)	*nodes*	1,147K	257K	218K (244K)	37,309	14,686
	elim		24.1	592.6 (583.36)	651.6	655.7
	cpu	591	– (–)	555 (261.67)	404	288
*scen*11-*f*4	*mem*	29		639 (196)	113	93
($\#V = 680$)	*nodes*	3,458K		365K (924K)	148K	125K
	elim			586.6 (593.1)	651.7	655.0

ory consumption: due to the size of the IPS_P, two instances remain unsolved. Combining ρ^{uni} with ρ^{exp} (i.e. ρ^{unex}) allows to save memory (between brackets, we have the results for ρ^{red} combined with ρ^{exp}). However, the best performance is obtained when combining explanation-based and proof-based reasoning, i.e. with ρ^{prex}. Note that the average size of the inconsistent partial states recorded in the base is very small: they involve about $680 - 655 = 25$ variables.

To summarize, the results that we have obtained with the new extraction operators and the dominance detection approach outperform, both in space and time, those obtained with the operator ρ^{red} which is dedicated to equivalence detection (ρ^{red} combined with ρ^{exp} gives similar results as ρ^{red} alone, except a memory reduction for a few instances). Besides, it allowed to solve more instances in a reasonable amount of time. We believe the results can still be improved since we did not control the partial states recorded in the base (and this has a clear impact when the resolution is difficult, as e.g. for the instance *scen*11-*f*4).

9 Conclusion

In this paper, we have introduced two operators that enable the extraction of an (inconsistent) partial state at each node of a search tree. Whereas the former collects information above the current node (propagation analysis from the root to the node) to perform an explanation-based extraction, the latter collects it below (subtree analysis) to perform a proof-based extraction – making these two approaches complementary. Next, we have shown that inconsistent partial states can be efficiently exploited to prune the search space by dominance detection.

State-based search as studied in [17] and in this paper can be seen as an approach to automatically break some form of local symmetries. A direct perspective of this new paradigm is to combine it with SBDD (Symmetry Breaking by Dominance Detection) [9, 10, 20, 21].

10 Acknowledgments

This paper has been supported by the CNRS and the ANR "Planevo" project n^{o}JC05_41940.

References

1. R.R. Baker, F. Dikker, F.Tempelman, and P.M. Wognum. Diagnosing and solving over-determined constraint satisfaction problems. In *Proceedings of IJCAI'93*, pages 276–281, 1993.
2. Christian Bessière. Arc-consistency in dynamic constraint satisfaction problems. In *Proceedings of AAAI'91*, pages 221–226, 1991.
3. F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.
4. K. Boutaleb, P. Jégou, and C. Terrioux. (no)good recording and robdds for solving structured (v)csps. In *Proceedings of ICTAI'06*, pages 297–304, 2006.
5. J.L. de Siqueira and J.F. Puget. Explanation-based generalisation of failures. In *Proceedings of ECAI'88*, pages 339–344, 1988.
6. R. Debruyne and C. Bessiere. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14:205–230, 2001.
7. R. Dechter and D. Frost. Backjump-based backtracking for constraint satisfaction problems. *Artificial Intelligence*, 136:147–188, 2002.
8. N. Eén and N. Sorensson. An extensible sat-solver. In *Proc. of SAT'03*, 2003.
9. T. Fahle, S. Schamberger, and M. Sellman. Symmetry breaking. In *Proceedings of CP'01*, pages 93–107, 2001.
10. F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proceedings of CP'01*, pages 77–92, 2001.
11. D. Frost and R. Dechter. Dead-end driven learning. In *Proceedings of AAAI'94*, pages 294–300, 1994.
12. M. Ginsberg. Dynamic backtracking. *Artificial Intelligence*, 1:25–46, 1993.
13. F. Hemery, C. Lecoutre, L. Sais, and F. Boussemart. Extracting MUCs from constraint networks. In *Proceedings of ECAI'06*, pages 113–117, 2006.
14. U. Junker. QuickXplain: preferred explanations and relaxations for over-constrained problems. In *Proceedings of AAAI'04*, pages 167–172, 2004.
15. G. Katsirelos and F. Bacchus. Generalized nogoods in CSPs. In *Proceedings of AAAI'05*, pages 390–396, 2005.
16. C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Recording and minimizing nogoods from restarts. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 1:147–167, 2007.
17. C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Transposition Tables for Constraint Satisfaction. In *Proceedings of AAAI'07*, pages 243–248, 2007.
18. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of DAC'01*, pages 530–535, 2001.
19. P. Prosser. Hybrid algorithms for the constraint satisfaction problems. *Computational Intelligence*, 9(3):268–299, 1993.
20. J.F. Puget. Symmetry breaking revisited. *Constraints*, 10(1):23–46, 2005.
21. M. Sellmann and P. Van Hentenryck. Structural symmetry breaking. In *Proceedings of IJCAI'05*, pages 298–303, 2005.
22. L. Zhang, C.F. Madigan, M.W. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proceedings of ICCAD'01*, pages 279–285, 2001.