Transposition Tables for Constraint Satisfaction

Christophe Lecoutre and **Lakhdar Sais** and **Sébastien Tabary** and **Vincent Vidal**
CRIL−CNRS FRE 2499
Université d'Artois
Lens, France
$\{lecoutre, sais, tabary, vidal\}$@cril.univ-artois.fr

Abstract

In this paper, a state-based approach for the Constraint Satisfaction Problem (CSP) is proposed. The key novelty is an original use of state memorization during search to prevent the exploration of similar subnetworks. Classical techniques to avoid the resurgence of previously encountered conflicts involve recording conflict sets. This contrasts with our state-based approach which records subnetworks – a snapshot of some selected domains – already explored. This knowledge is later used to either prune inconsistent states or avoid recomputing the solutions of these subnetworks. Interestingly enough, the two approaches present some complementarity: different states can be pruned from the same partial instantiation or conflict set, whereas different partial instantiations can lead to the same state that needs to be explored only once. Also, our proposed approach is able to dynamically break some kinds of symmetries (e.g. neighborhood interchangeability). The obtained experimental results demonstrate the promising prospects of state-based search.

Introduction

In classical heuristic search algorithms (A*, IDA*, ...) or game search algorithms (α-β, SSS* ...), where nodes in the search tree represent world states and transitions represent moves, many states may be encountered several times at possibly different depths. This is due to the fact that different sequences of moves from the initial state of the problem can yield identical situations of the world. Moreover, a state S in a node at depth i of the search tree cannot lead to a better solution than a node containing the same state S previously encountered at a depth $j < i$. As a consequence, some portions of the search space may be unnecessarily evaluated and explored several times, which may be costly.

The phenomenon of revisiting identical states reached from different sequences of transitions, better known as *transpositions*, has been identified very early in the context of chess software (Greenblatt, Eastlake, & Crocker 1967; Slate & Atkin 1977; Marsland 1992). One solution to this problem is to store the encountered nodes, plus some related information (e.g. depth, heuristic evaluation), in a so-called *transposition table*. The data structure used to implement such a transposition table is classically a hash table whose key is computed from the description of the state, such as the key based on the logical XOR operator for chess (Zobrist 1970). The amount of memory in a machine being limited, the number of entries in a transposition table may be bounded. This technique has been adapted to heuristic search algorithms such as IDA* (Reinefeld & Marsland 1994), and is also successfully employed in modern automated STRIPS planners such as e.g. FF (Hoffmann & Nebel 2001) and YAHSP (Vidal 2004).

A direct use of transposition tables in complete CSP backtracking search algorithms is clearly useless, one property of this kind of algorithm being that the state of a constraint network (variables, constraints and reduced domains) cannot be encountered twice. Indeed, with a binary branching scheme for example, once it has been proven that a positive decision $X = v$ leads to a contradiction, the opposite decision $X \neq v$ is immediately taken in the other branch. In other words, in the first branch the domain of X is reduced to the singleton $\{v\}$, while in the second branch v is removed from the domain of X: obviously, no state where $X = v$ has been asserted can be identical to a state where $X \neq v$ is true.

However, we show in this paper that two different sequences of decisions performed during the resolution of a constraint network P can lead to two networks P_1 and P_2 that can be reduced to the same *subnetwork*. We exhibit three *reduction operators* which, in that case, satisfy the following property: P_1 is satisfiable if and only if P_2 is satisfiable. These operators remove some selected variables and constraints involving them. For example, depending on the operator, removed variables belong to the scope of universal constraints or have an associated domain that has not been filtered yet. In practice, once a subnetwork has been identified, it can simply be stored into a transposition table that will be checked before expanding each node. A lookup in the table will then avoid to explore a network that can be reduced to a subnetwork already encountered.

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 $vars(P)$. 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 is the set of instantiations allowed for the variables of its scope. 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 if $sol(P) \neq \emptyset$.

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 (e.g. see (Bessiere 2006)). It is for example maintained during search by the algorithm MGAC, called MAC in the binary case.

From now on, we will consider an inference operator ϕ that enforces a domain filtering consistency (Debruyne & Bessiere 2001) and can be employed at any step of a tree search. For a constraint network P and 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 $dom(X)$. $\phi(P)$ 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 $\phi(P) = \perp$.

A constraint is *universal* if every valid instantiation built from the current domains of its variables satisfies it.

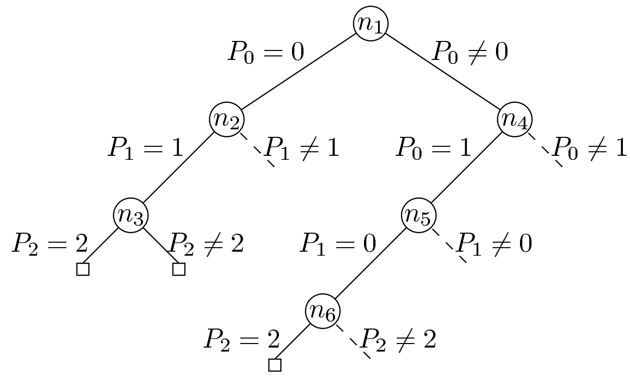
Definition 1 *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN. A constraint $C \in \mathscr{C}$ with $scp(C) = \{X_1, \ldots, X_r\}$ is universal if $\forall v_1 \in dom(X_1), \ldots, \forall v_r \in dom(X_r), \exists t \in rel(C)$ such that $t[X_1] = v_1, \ldots, t[X_r] = v_r$.*

A *constraint subnetwork* can be obtained from a CN by removing a subset of its variables and the constraints involving them.

Definition 2 *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN and $S \subseteq \mathscr{X}$. The constraint subnetwork $P \ominus S$ is the CN $(\mathscr{X}', \mathscr{C}')$ such that $\mathscr{X}' = \mathscr{X} \setminus S$ and $\mathscr{C}' = \{C \in \mathscr{C} \mid scp(C) \cap S = \emptyset\}$.*

Illustration

Let us now illustrate our purpose with the classical *pigeon holes* problem with five pigeons. This problem involves five variables $P_0, \ldots, P_4$ that represent the pigeons, whose domains initially equal to $\{0, \ldots, 3\}$ represent the holes. The constraints state that two pigeons cannot be in the same hole, making this problem unsatisfiable as there are five pigeons for only four holes. They may be expressed with a clique of binary constraints: $P_0 \neq P_1, P_0 \neq P_2, \ldots, P_1 \neq P_2, \ldots$ Figure 1 depicts a partial view of a search tree for this problem, built by MAC with a binary branching scheme, and the state of the domains at each node.



$$n_1{:}\begin{bmatrix} P_0 : \{0,1,2,3\} \\ P_1 : \{0,1,2,3\} \\ P_2 : \{0,1,2,3\} \\ P_3 : \{0,1,2,3\} \\ P_4 : \{0,1,2,3\} \end{bmatrix} \quad n_2{:}\begin{bmatrix} P_0 : \{0\} \\ P_1 : \{1,2,3\} \\ P_2 : \{1,2,3\} \\ P_3 : \{1,2,3\} \\ P_4 : \{1,2,3\} \end{bmatrix} \quad n_3{:}\begin{bmatrix} P_0 : \{0\} \\ P_1 : \{1\} \\ \mathbf{P_2 : \{2,3\}} \\ \mathbf{P_3 : \{2,3\}} \\ \mathbf{P_4 : \{2,3\}} \end{bmatrix}$$

$$n_4{:}\begin{bmatrix} P_0 : \{1,2,3\} \\ P_1 : \{0,1,2,3\} \\ P_2 : \{0,1,2,3\} \\ P_3 : \{0,1,2,3\} \\ P_4 : \{0,1,2,3\} \end{bmatrix} \quad n_5{:}\begin{bmatrix} P_0 : \{1\} \\ P_1 : \{0,2,3\} \\ P_2 : \{0,2,3\} \\ P_3 : \{0,2,3\} \\ P_4 : \{0,2,3\} \end{bmatrix} \quad n_6{:}\begin{bmatrix} P_0 : \{1\} \\ P_1 : \{0\} \\ \mathbf{P_2 : \{2,3\}} \\ \mathbf{P_3 : \{2,3\}} \\ \mathbf{P_4 : \{2,3\}} \end{bmatrix}$$

Figure 1: Pigeon holes: partial search tree

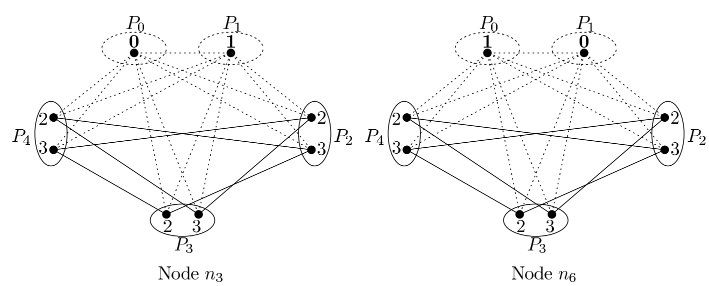


Figure 2: Pigeon holes: two similar subnetworks

We can first remark that the six nodes $n_1, \ldots, n_6$ represent networks that are all different, as the domains of the variables differ by at least one value. However, let us focus on nodes n_3 and n_6. The only difference lies in the domains of P_0 and P_1, which are respectively reduced to the singletons $\{0\}$ and $\{1\}$ in n_3, and $\{1\}$ and $\{0\}$ in n_6. The domains of the other variables P_2, P_3 and P_4 are all equal to $\{2, 3\}$. The networks associated to nodes n_3 and n_6 are represented in Figure 2, including all instantiations allowed for each constraint. We can see that the structure of these networks is very similar, the only difference being the inversion of the values 0 and 1 between P_0 and P_1.

The two crucial points about n_3 and n_6 are the following: (1) neither P_0 nor P_1 will play a role in subsequent search anymore, and (2) checking the satisfiability of (the network attached to node) n_3 is equivalent to check the satisfiability of n_6. Point (1) is easy to see: as arc consistency (AC) is maintained, all constraints involving P_0 and P_1 are universal: whatever is the assignment of a value to the other variables, these constraints will be satisfied. Variables P_0 and

P_1 can thus be disconnected from the constraint networks in n_3 and n_6. As a consequence, we can immediately see that point (2) is true: the constraint subnetworks consisting of the remaining variables P_2, P_3 and P_4 and the constraints involving them are equal, and as a consequence n_3 is satisfiable if and only if n_6 is satisfiable. Then, if we had stored that subnetwork in a transposition table after proving the unsatisfiability of n_3, we could have avoided expanding n_6 by a simple lookup in the table.

Identifying Constraint Subnetworks

Storing complete states in a transition table is not relevant, as they cannot be encountered twice by a complete CSP search algorithm such as MGAC. We have then to address the problem of identifying constraint subnetworks, that may be reached several times during a tree search. Such subnetworks should ideally be as general as possible while just requiring a small amount of memory. We define in this section three reduction operators whose objective is essentially to minimize the number of recorded variables. Intuitively, the fewer the number of recorded variables is, the higher the pruning capability is. This also contributes to reduce memory consumption.

Preserving Solutions (operator ρ^{sol})

The first operator, denoted ρ^{sol}, preserves the set of solutions of a given CN. Applied to a network, it consists in removing so-called *s-eliminable* variables which have a singleton domain and only appear in universal constraints.

Definition 3 *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN. A variable $X \in \mathscr{X}$ is s-eliminable if $|dom(X)| = 1$ and $\forall C \in \mathscr{C} \mid X \in scp(C)$, C is universal. The set of s-eliminable variables of P is denoted by $S_{elim}(P)$.*

The operator ρ^{sol} removes all s-eliminable variables from a CN as well as all constraints involving at least one of them.

Definition 4 *Let P be a CN. $\rho^{sol}(P) = P \ominus S_{elim}(P)$.*

The solutions of a network P can be enumerated by expanding the solutions of the subnetwork $\rho^{sol}(P)$ with the interpretation built from the s-eliminable variables. Indeed, domains of removed variables are singleton, and removed constraints have no more impact on the network.

Proposition 1 *Let P be a CN and $t = \{(X, v) \mid X \in S_{elim}(P) \wedge dom(X) = \{v\}\}$. $sol(P) = \{s \cup t \mid s \in sol(\rho^{sol}(P))\}$.*

Proof. Obviously, any solution of P also satisfies all constraints of $\rho^{sol}(P)$. The converse is immediate: any solution s of $\rho^{sol}(P)$ can be extended to a unique solution of P since eliminated variables (of $S_{elim}(P)$) have a singleton domain and eliminated constraints are universal. □

Preserving Satisfiability (operator ρ^{uni})

The second operator, denoted ρ^{uni}, preserves the satisfiability of a given CN (but not necessarily all solutions). Applied to a network, it removes so-called *u-eliminable* variables. Such variables only appear in universal constraints.

Definition 5 *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN. A variable $X \in \mathscr{X}$ is u-eliminable if $\forall C \in \mathscr{C} \mid X \in scp(C)$, C is universal. The set of u-eliminable variables of P is denoted by $U_{elim}(P)$.*

The operator ρ^{uni} removes u-eliminable variables from a CN as well as constraints involving at least one of them.

Definition 6 *Let P be a CN. $\rho^{uni}(P) = P \ominus U_{elim}(P)$.*

Satisfiability is preserved by ρ^{sol} since eliminated constraints have no more impact on the network (as they are universal). It is clear that for any CN P, $\rho^{uni}(P)$ is a subnetwork of $\rho^{sol}(P)$ since s-eliminable variables are also u-eliminable. Hence, ρ^{sol} can be considered as a special case of ρ^{uni}.

Proposition 2 *A constraint network P is satisfiable if and only if $\rho^{uni}(P)$ is satisfiable.*

Proof. Removing any universal constraint does not change the satisfiability of a network. Here, we only remove universal constraints and variables that become disconnected from the network. □

Reducing Subnetworks (operator ρ^{red})

The third operator, denoted ρ^{red}, extracts a reduced subnetwork from a current network by removing both u-eliminable variables and so-called *r-eliminable* variables. The latter correspond to variables whose domain remains unchanged after taking a set of decisions and applying an inference operator.

Definition 7 *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN, ϕ an inference operator, Δ a set of decisions and $P' = \phi(P|_\Delta)$. A variable $X \in \mathscr{X}$ is r-eliminable in P' w.r.t. P if $dom^{P'}(X) = dom^{P}(X)$. The set of r-eliminable variables of P' is denoted by $R^{P}_{elim}(P')$.*

Definition 8 *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN, ϕ an inference operator, Δ a set of decisions and $P' = \phi(P|_\Delta)$. $\rho^{red}(P') = \rho^{uni}(P') \ominus R^{P}_{elim}(P')$.*

The main result of this paper states that, when two networks derived from a given network can be reduced to the same subnetwork, the satisfiability of one determines the satisfiability of the other one.

Proposition 3 *Let P be a CN, ϕ an inference operator, and Δ_1, Δ_2 two sets of decisions. Let $P_1 = \phi(P|_{\Delta_1})$ and $P_2 = \phi(P|_{\Delta_2})$. If $\rho^{red}(P_1) = \rho^{red}(P_2)$, then P_1 is satisfiable if and only if P_2 is satisfiable.*

Proof. Let us show that $sol(P_1) \neq \emptyset \Rightarrow sol(P_2) \neq \emptyset$. From any solution $s \in sol(P_1)$, it is possible to build a solution $s' \in sol(P_2)$ as follows: if $s[X] \in dom^{P_2}(X)$ then $s'[X] = s[X]$, otherwise $s'[X] = a$ where a is any value taken in $dom^{P_2}(X)$. For any constraint C of P, we know that s satisfies C. Let us show now that s' also satisfies C. It is clear that this is the case if $\nexists X \in scp(C) \mid s[X] \neq s'[X]$. Next, if $\exists X \in scp(C) \mid s[X] \neq s'[X]$, we can show that C is universal, and consequently, C is satisfied by s'. Indeed, we show below that $s'[X] \neq s[X]$ implies $X \in U_{elim}(P_2)$ from which we can deduce that any constraint involving

X is universal (by definition of U_{elim}). Let us suppose that $X \in vars(\rho^{uni}(P_2))$ (and then, $X \notin U_{elim}(P_2)$). If $X \in R^P_{elim}(P_2)$ then it is immediate that $s[X] \in dom^{P_2}(X)$ (and so, $s'[X] \neq s[X]$ is impossible). Otherwise, it means that X belongs to $\rho^{red}(P_2)$, and consequently also belongs to $\rho^{red}(P_1)$. As the domains of the variables of these two subnetworks are equal (by hypothesis), by construction of s', we cannot have $s'[X] \neq s[X]$. So, we can conclude that $X \in U_{elim}(P_2)$. Finally, we can apply the same reasoning to show that $sol(P_2) \neq \emptyset \Rightarrow sol(P_1) \neq \emptyset$. □

Proposition 4 *Let P be a binary CN, Δ a set of decisions, and $P' = (\mathscr{X}', \mathscr{C}') = \rho^{red}(\mathrm{AC}(P|_\Delta))$. We have then: $\forall X \in \mathscr{X}', 1 < |dom^{P'}(X)| < |dom^P(X)|$.*

Proof. A variable X with a singleton domain is removed since all constraints involving X are universal. Indeed, as arc consistency (AC) is enforced, all values for any variable Y connected to X (constraints being binary) are compatible with the single value of X. By definition of ρ^{red}, a variable X such that $|dom^{P'}(X)| = |dom^P(X)|$ is removed. □

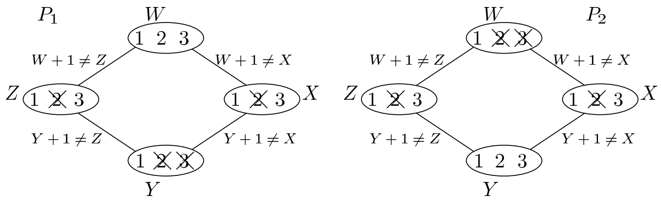
Pruning Capability of Reduction Operators

In the context of checking the satisfiability of a CSP instance, we discuss now the pruning capability of the operators we defined. Clearly, the more variables are removed from a network P_1 to produce a subnetwork P'_1, the more networks will be discarded by P'_1. Indeed, if a network P_2 can be discarded because it can be reduced to P'_1, then the domains of the variables of P_2 that do not appear in P'_1 can be in any state: they either contain all the values w.r.t. the initial problem, or are reduced in such a way that they belong to universal constraints only. It is easy to see that, by definition of the reduction operators, $vars(\rho^{red}(P)) \subseteq vars(\rho^{uni}(P)) \subseteq vars(\rho^{sol}(P))$. We can then expect ρ^{red} to have a better (at least equal) pruning capability than ρ^{sol} and ρ^{uni}.

This is illustrated by the networks depicted in Figure 3. On the first one, the assignment $Y = 1$ on an initial network P (where all domains are $\{1, 2, 3\}$) leaves the domain of W unchanged and eliminates the value 2 from both $dom(X)$ and $dom(Z)$, yielding the derived network $P_1 = \mathrm{AC}(P|_{Y=1})$. On the second one, the assignment $W = 1$ on P leaves the domain of Y unchanged and eliminates the value 2 from both $dom(X)$ and $dom(Z)$, yielding the derived network $P_2 = \mathrm{AC}(P|_{W=1})$. Whereas ρ^{uni} produces two different subnetworks from P_1 and P_2, ρ^{red} applied to them leads to the same reduced subnetwork, whatever is the reason of variable elimination (u-eliminable or r-eliminable). As a consequence, we know from Proposition 3 that, once P_1 or P_2 has been explored, the exploration of the other one is useless to determine its satisfiability.

Search Algorithm Exploiting ρ^{red}

In this section, we succinctly present an algorithm that performs a depth-first search, maintains a domain filtering consistency ϕ (at least, checking that constraints only involving singleton-domain variables are satisfied) and prunes inconsistent states using ρ^{red}. The main idea is to record



$$\rho^{uni}(P_1) = \begin{bmatrix} W : \{1,2,3\} \\ X : \{1,3\} \\ Z : \{1,3\} \end{bmatrix} \qquad \rho^{uni}(P_2) = \begin{bmatrix} X : \{1,3\} \\ Y : \{1,2,3\} \\ Z : \{1,3\} \end{bmatrix}$$

$$\rho^{red}(P_1) = \begin{bmatrix} X : \{1,3\} \\ Z : \{1,3\} \end{bmatrix} \qquad \rho^{red}(P_2) = \begin{bmatrix} X : \{1,3\} \\ Z : \{1,3\} \end{bmatrix}$$

Figure 3: Network reduction by ρ^{uni} and ρ^{red}

in a transposition table all reduced subnetworks extracted from nodes that have been proven inconsistent. Then, nodes whose extracted reduced subnetwork already belongs to the transposition table can be safely pruned.

The recursive function $solve$ determines the satisfiability of a network P (see Algorithm 1). At a given stage, if the current network (after enforcing ϕ) is inconsistent, $false$ is returned (line 2) whereas if all variables have been assigned, $true$ is returned (line 3). Otherwise, we check if the current node can be pruned thanks to the transposition table (line 4). If search continues, we select a pair (X, a) and recursively call $solve$ by considering two branches, one labelled with $X = a$ and the other with $X \neq a$ (lines 5 and 6). If a solution is found, $true$ is returned. Otherwise, the current network has been proven inconsistent and its reduced subnetwork is added to the transposition table (line 7) before returning $false$.

This algorithm could be slightly modified to enumerate all the solutions of a network by using ρ^{sol}. To do that, the transposition table should store all encountered subnetworks (not only the unsatisfiable ones), along with an additional information: their solution set. When a network P is such that $\rho^{sol}(P)$ already belongs to the table, the solutions of P can be expanded from the solutions of $\rho^{sol}(P)$ stored into the table, with the interpretation built from the s-eliminable variables of P (c.f. Proposition 1). Similarly, to count the number of solutions, one can associate to each reduced subnetwork stored in the table, the number of its solutions.

Algorithm 1 $solve(P_{init} = (\mathscr{X}, \mathscr{C})$: CN) : Boolean

1: $P = \phi(P_{init})$
2: **if** $P = \bot$ **then** return $false$
3: **if** $\forall X \in \mathscr{X}, |dom(X)| = 1$ **then** return $true$
4: **if** $\rho^{red}(P) \in transposition\_table$ **then** return $false$
5: select a pair (X, a) with $|dom(X)| > 1 \wedge a \in dom(X)$
6: **if** $solve(P|_{X=a})$ or $solve(P|_{X \neq a})$ **then** return $true$
7: add $\rho^{red}(P)$ to $transposition\_table$
8: return $false$

State-Based Search: Scope and Relationships

The scope of our approach is related to several key issues in constraint programming. Indeed, state based search is able to automatically eliminate some kinds of symmetries during

		brelaz		*dom/wdeg*	
		$\neg SBS$	SBS	$\neg SBS$	SBS
Pigeons-11	*cpu*	265.48	2.33	272.73	6.35
	nodes	$4,421K$	5,065	$4,441K$	61,010
	hits	0	4,008	0	40,014
Pigeons-13	*cpu*	*timeout*	4.57	*timeout*	26.44
	nodes	–	24,498	–	$327K$
	hits	–	20,350	–	$245K$
Pigeons-15	*cpu*	*timeout*	12.66	*timeout*	81.58
	nodes	–	$115K$	–	$900K$
	hits	–	98,124	–	$728K$
Pigeons-18	*cpu*	*timeout*	116.19	*timeout*	*timeout*
	nodes	–	$1,114K$	–	–
	hits	–	$983K$	–	–

Table 1: Cost of running MAC without and with SBS on Pigeon Hole instances

		dom/wdeg	
		$\neg SBS$	SBS
scen11-f8	*cpu*	14.84	15.74
	nodes	15,045	13,858
	hits	0	370
scen11-f7	*cpu*	57.68	15.36
	nodes	$113K$	14,265
	hits	0	919
scen11-f6	*cpu*	110.18	18.67
	nodes	$217K$	18,938
	hits	0	1252
scen11-f5	*cpu*	550.55	162.32
	nodes	$1,147K$	$257K$
	hits	0	17,265

Table 2: Cost of running MAC without and with SBS on hard RLFAP instances

search, and presents strong complementarity with nogood recording.

Neighborhood interchangeability is a weak form of (full) interchangeability (Freuder 1991) that can be exploited in practice to reduce the search space. Given a variable X, two values a and b in $dom(X)$ are neighborhood interchangeable if for any constraint C involving X, the set of supports of a for X in C is equal to the set of supports of b for X in C. We can observe that our state-based approach discards redundant states coming from interchangeable values. Indeed, if P is a network such that values a and b for a variable X of P are interchangeable, it clearly appears that the subnetworks extracted from $P|_{X=a}$ and $P|_{X=b}$ are identical after applying any ρ operator.

Interchangeability is related to symmetry (Cohen *et al.* 2006) whose objective is to discard parts of the search tree that are symmetrical to already explored subtrees. This can lead to a dramatic reduction of the search effort required to solve a constraint network. To reach this goal, one has first to identify symmetries and then, to exploit them. Different approaches have been proposed to exploit symmetries; the most related one being symmetry breaking via dominance detection (SBDD).

The principle of SBDD is the following: every time the search algorithm reaches a new node, one just checks whether this node is equivalent to or dominated by a node that has already been expanded earlier. This approach requires (1) the memorization of information about nodes explored during search (2) the exploitation of this information by considering part or all of the symmetries from the symmetry group associated with the initial network. The information stored for a node can be the current domains of all variables at the node, called Global Cut Seed in (Focacci & Milano 2001) and pattern in (Fahle, Schamberger, & Sellman 2001). But it can also be reduced to the set of decisions labelling the path from the root to the node (Puget 2005). In our case, we only store the current domains of a subset of variables (for ρ^{red}, those that are neither u-eliminable nor r-eliminable) of the initial network, which allows us to automatically break some kinds of local symmetry. Interestingly, we can imagine to combine the two approaches, using the general "nogoods" extracted by our method with dominance detection via a set of symmetries.

Finally, we discuss the complementarity between state based search and nogood recording (e.g. see (Dechter 1990)). On the one hand, a given state corresponding to a subnetwork already shown unsatisfiable represents in a compact way an exponential number of nogoods. On the other hand, a given (minimal) nogood represents an exponential number of instantiations. The complementarity of these two paradigms appears in their ability to avoid redundant search. Indeed, a given nogood avoids (or cuts) several states, whereas a given state cuts several partial instantiations i.e. instantiations leading to the same state.

Experiments

In order to show the practical interest of state-based search, we have conducted an experimentation on benchmarks from the second CSP solver competition (`http://cpai.ucc.ie/06/Competition.html`) on a PC Pentium IV 2.4GHz 1024Mb under Linux. We have used the algorithm MGAC, and studied the impact of state-based search, denoted SBS, with various variable ordering heuristics. Performance is measured in terms of number of visited nodes (nodes), cpu time in seconds (cpu) and number of discarded nodes by SBS (hits). Remark that SBS can be applied to constraints defined in extension or in intention (but, according to the selected reduction operator, dealing with global constraints may involve some specific treatment).

We have implemented Algorithm 1, but have considered a subset of U_{elim}, as determining u-eliminable variables involved in non binary constraints grows exponentially with the arity of the constraints. More precisely, in our implementation, the ρ^{uni} operator (called by ρ^{red}) only removes the variables with a singleton domain involved in constraints binding at most one non singleton-domain variable. Computing this restricted set can be done in linear time. For binary networks, any variable with a singleton domain is automatically removed by our operator (see Proposition 4). The transposition table used to store the subnetworks is implemented as a hash table whose key is the concatenation of couples (id, dom) where id is a unique integer associated with each variable and dom the domain of the variable itself represented as a bit vector. To search one solution only, no additional data needs to be stored in an entry of the table, as the presence of a key is sufficient to discard a node.

Table 1 presents results obtained on some $pigeon\ hole$ instances. One can observe the interest of SBS on this problem since many nodes can be discarded. Here, one can note that it is more interesting to use the heuristic $brelaz$ (identical results are obtained with $dom/ddeg$ (Bessiere & Régin 1996)) than $dom/wdeg$ (Boussemart *et al.* 2004). This can

		brelaz		*dom/ddeg*		*dom/wdeg*	
Instances		$\neg SBS$	SBS	$\neg SBS$	SBS	$\neg SBS$	SBS
composed-25-10-20-4-ext	*cpu*	179.84	4.27	2.82	2.6	2.7	2.57
	nodes (hits)	$1,771K$	$9,944\ (2,609)$	$1,644$	$784\ (24)$	262	$255\ (5)$
composed-25-10-20-9-ext	*cpu*	857.07	3.73	12.13	10.25	2.58	2.66
	nodes (hits)	$10M$	$7,935\ (1,738)$	$75,589$	$54,245\ (2,486)$	323	$323\ (0)$
dubois-21-ext	*cpu*	*timeout*	480.98	*timeout*	384.84	911.51	295.81
	nodes (hits)	–	$6,292K\ (2,097K)$	–	$4,194K\ (2,097K)$	$16M$	$3,496K\ (1,573K)$
dubois-22-ext	*cpu*	*timeout*	*timeout*	*timeout*	*timeout*	*timeout*	527.19
	nodes (hits)	–	–	–	–	–	$6,641K\ (3,146K)$
pret-60-25-ext	*cpu*	687.31	5.65	471.16	5.82	416.49	2.27
	nodes (hits)	$12M$	$55,842\ (13,188)$	$7,822K$	$47,890\ (13,188)$	$7,752K$	$4,080\ (1,384)$
pret-150-25-ext	*cpu*	*timeout*	*timeout*	*timeout*	*timeout*	*timeout*	10.34
	nodes (hits)	–	–	–	–	–	$97,967\ (37,457)$

Table 3: Cost of running MGAC without and with SBS on structured instances

be explained by the fact that the former is closer to the lexicographic order which is well adapted for this problem.

In Table 2, we focus on some difficult real-world instances of the Radio Link Frequency Assignment Problem (RLFAP). Even with SBS, these instances cannot be solved within $1,200$ seconds when using $brelaz$ or $dom/ddeg$, so we only present the results obtained with $dom/wdeg$. We can see about a 4-fold improvement when using SBS. In Table 3, we can see the results obtained for some binary and non binary instances (dubois and pret instances involve ternary constraints) for which our approach is effective.

We can summarize the results of our experimentation as follows. When SBS is ineffective, the solver is slowed down by approximately 15%. Yet, by analysing the behaviour of SBS, one can decide at any time to stop using it and free memory: the cpu time lost by the solver is then bounded by the time alloted to the analysis. When SBS is effective, and this is the case on some series, the improvement can be very significant in cpu time and number of solved instances.

One can wonder about the amount of memory required to store the different states. An interesting thing is that as u-eliminable and r-eliminable variables are removed, a lot of space can be saved, in particular on sparse constraint graphs. For instance, only $265MiB$ were necessary to record the $50,273$ different subnetworks when solving (in 162 seconds) the large instance $scen11$-$f5$ that involves 680 variables with domains up to 39 values.

Conclusion

In this paper, we provided the proof of concept of the exploitation, for constraint satisfaction, of a well-known technique widely used in search: pruning from transpositions. This has not been addressed so far since, in CSP, contrary to search, two branches of a search tree cannot lead to the same state. This led us to define some reduction operators that keep partial information from a node, sufficient to detect constraint networks that do not need to be explored.

We actually addressed the theoretical and practical aspects of how to exploit these operators in terms of equivalence between nodes. Two immediate prospects of this work concern the definition of more powerful reduction operators and the exploitation of dominance properties between nodes. Also, many links with the concept of symmetry have still to be investigated, and we can expect a cross-fertilization between state-based search and symmetry breaking methods.

Acknowledgments

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

References

Bessiere, C., and Régin, J. 1996. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings of CP'96*, 61–75.

Bessiere, C. 2006. Constraint propagation. In *Handbook of Constraint Programming*. Elsevier. chapter 3.

Boussemart, F.; Hemery, F.; Lecoutre, C.; and Sais, L. 2004. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, 146–150.

Cohen, D.; Jeavons, P.; Jefferson, C.; Petrie, K.; and Smith, B. 2006. Symmetry definitions for constraint satisfaction problems. *Constraints* 11(2-3):115–137.

Debruyne, R., and Bessiere, C. 2001. Domain filtering consistencies. *Journal of Artificial Intelligence Research* 14:205–230.

Dechter, R. 1990. Enhancement schemes for constraint processing: backjumping, learning and cutset decomposition. *Artificial Intelligence* 41:273–312.

Fahle, T.; Schamberger, S.; and Sellman, M. 2001. Symmetry breaking. In *Proceedings of CP'01*, 93–107.

Focacci, F., and Milano, M. 2001. Global cut framework for removing symmetries. In *Proceedings of CP'01*, 77–92.

Freuder, E. 1991. Eliminating interchangeable values in constraint satisfaction problems. In *Proc. of AAAI'91*, 227–233.

Greenblatt, R.; Eastlake, D.; and Crocker, S. 1967. The Greenblatt chess program. In *Proceedings of Fall Joint Computer Conference*, 801–810.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Marsland, T. 1992. Computer chess and search. In *Encyclopedia of Artificial Intelligence*. J. Wiley & Sons. 224–241.

Puget, J. 2005. Symmetry breaking revisited. *Constraints* 10(1):23–46.

Reinefeld, A., and Marsland, T. A. 1994. Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16(7):701–710.

Slate, D., and Atkin, L. 1977. Chess 4.5: The northwestern university chess program. In *Chess Skill in Man and Machine*. Springer Verlag. 82–118.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proceedings of ICAPS'04*, 150–159.

Zobrist, A. L. 1970. A new hashing method with applications for game playing. Technical Report 88, Computer Sciences Dept., Univ. of Wisconsin.