

# Generalizing Partial Order and Dynamic Backtracking

Christian Bliet

Artificial Intelligence Laboratory  
Swiss Federal Institute of Technology  
IN-Ecublens 1015 Lausanne Switzerland  
cbliet@lia.di.epfl.ch

## Abstract

Recently two new backtracking algorithms *dynamic backtracking* (DB) and *partial order dynamic backtracking* (PDB) have been presented. These algorithms have the property to be additive on disjoint subproblems and yet use only polynomial space. Unlike DB PDB only imposes a partial search order and therefore appears to have more freedom than DB to explore the search space. However both algorithms are not directly comparable in terms of flexibility. In this paper we present new backtracking algorithms that are obtained by relaxing the ordering conditions of PDB. This gives them additional flexibility while still being additive on disjoint subproblems. In particular we show that our algorithms generalize both DB and PDB.

## Introduction

Most sound and complete algorithms for solving constraint satisfaction problems are based on backtracking. Intelligent backtrackers record information regarding dead ends to avoid encountering them again. This approach was first used in *dependency directed backtracking* (DDB) where the subset of the current assignments that caused the dead end are recorded in a nogood (Stallman & Sussman 1977). By avoiding this subset of assignments the efficiency of the subsequent search can be increased. Unfortunately since the number of accumulated nogoods increases monotonically DDB has an exponential space complexity.

To address this problem recent backtrackers eliminate nogoods that are no longer relevant to the current assignments. By doing so the space complexity remains polynomial. One of the standard techniques that has adopted this approach is *conflict based backjumping*<sup>1</sup> (CBJ) (Prosser 1993). When a dead end is encountered CBJ jumps back to the variable that participated in the dead end and was instantiated last. However by doing so CBJ erases all assignments and nogoods that were obtained since. The search effort to solve disjoint subproblems is therefore multiplicative not additive.

<sup>1</sup>This algorithm also appears in (Ginsberg 1993) where it is called *backjumping*.

To overcome this drawback *dynamic backtracking* (DB) reutilizes intermediate search information (Ginsberg 1993). Upon backtracking it only removes the assignment of the backtrack variable and the nogoods that depend on it. This gives DB the property to be additive on disjoint subproblems. However the forced reordering undoes the choices performed by variable selection heuristics. Experiments indicate that this negative effect often outweighs the benefits (Baker 1994).

CBJ and DB can be considered as classical backtrack search algorithms. The search consists in extending a consistent partial set of assignments until a solution is found. An alternative approach is to consider a complete set of assignments that is incrementally modified until all constraints are satisfied. During the search a partial order on the variables is imposed to guarantee termination. This approach was first introduced by *partial order backtracking* (POB) (McAllester 1993). In (Ginsberg & McAllester 1994) this algorithm is slightly generalized and is referred to as PDB. Like DB PDB has the property to be additive on disjoint subproblems.

The advantage of PDB with respect to DB is that it allows to reorder all variables and not just the “future” variables. It therefore appears to be much more flexible in terms of exploration of the search space. Nevertheless DB and PDB are not directly comparable in terms of flexibility. In fact in (McAllester 1993) it was raised as an open question whether a more general algorithm could be devised. In this paper we present two new intelligent backtrackers that are obtained by relaxing the ordering conditions of PDB. This gives them additional flexibility while still being additive on disjoint subproblems. In particular we show that our algorithms generalize both DB and PDB.

## Background

To avoid redundant search intelligent backtrackers record nogoods. A nogood  $\gamma$  is a subset of assignments of values  $v_j$  to variables  $x_j$  which are incompatible:

$$\neg(x_1 = v_1 \wedge \dots \wedge x_k = v_k) \quad (1)$$

DDB accumulates all nogoods it encounters and hence suffers from an exponential space complexity. To keep the space complexity polynomial some nogoods need to be removed during the search. In this paper we will follow the approach used by PDB which is outlined below.

We refer the reader to (McAllester 1993) and (Ginsberg & McAllester 1994) for a more detailed presentation.

At all times a complete set of assignments of the variables is considered. These assignments are incrementally modified until all constraints are satisfied. This search process is driven by the addition of new nogoods. New nogoods are added when the current assignments violate a constraint or when backtracking occurs. To satisfy a new nogood  $\gamma$  one of its variable assignments  $\Gamma$  say  $x_k$   $\Gamma$  needs to be changed. This is represented explicitly by rewriting expression (1) for  $\gamma$  as:

$$x_1 = v_1 \wedge \dots \wedge x_{k-1} = v_{k-1} \rightarrow x_k \neq v_k$$

We will refer to this expression as the ordered nogood  $\tilde{\gamma}$ . Here  $x_1 = v_1$  through  $x_{k-1} = v_{k-1}$  are called the *antecedent* and  $x_k \neq v_k$  the *conclusion* of  $\tilde{\gamma}$ . As soon as the value of  $x_k$  changes  $\Gamma$  all nogoods that have  $x_k$  as antecedent variable are removed. This means that the current assignments match all the antecedents of each nogood. Note that some assignments do not appear in any nogood antecedent. If desired  $\Gamma$  these assignments may be changed in addition to the one of  $x_k$ . We therefore say that an assignment is *acceptable* for a set of nogoods if all the antecedents are matched and none of the conclusions are violated. Observe that with this approach  $\Gamma$  a variable assignment can be ruled out by at most one nogood. There are therefore at most  $nd$  nogoods  $\Gamma$  where  $n$  is the number of variables and  $d$  the maximum domain size. Since each nogood requires  $\mathcal{O}(n)$  space  $\Gamma$  the overall space complexity is  $\mathcal{O}(n^2d)$ .

Backtracking occurs when for a given variable  $\Gamma$  say  $y$   $\Gamma$  all values  $v_1$  through  $v_d$  are ruled out by nogoods  $\Sigma_i \rightarrow y \neq v_i$ . In this case a new nogood is generated by the following inference rule:

$$\frac{\begin{array}{l} \Sigma_1 \rightarrow y \neq v_1 \\ \vdots \\ \Sigma_d \rightarrow y \neq v_d \end{array}}{\neg(\Sigma_1 \wedge \dots \wedge \Sigma_d)} \quad (2)$$

New nogoods either correspond to constraint violations or are obtained by the above inference rule. This means that all steps in the inference process are valid. So when the empty nogood is obtained it can safely be concluded that the problem has no solution. It must be emphasized that this is true regardless of any considerations concerning variable ordering or nogood removal.

Each nogood eliminates a new section of the search space. Since DDB accumulates nogoods and the size of the search space is bounded  $\Gamma$  DDB will terminate. However  $\Gamma$  when nogoods are removed during the search  $\Gamma$  termination is no longer guaranteed. CBJ and DB therefore impose a total order on the assigned variables. By doing so  $\Gamma$  the portion of the search space ruled out by the new nogood includes the portions that were ruled out by the nogoods that are removed. As a result the size of the eliminated search space increases monotonically which ensures termination.

$\tilde{\gamma}$	$x_1$	$x_2$	$x_3$	$\Gamma$
	2	2	2	
$x_2 = 2 \rightarrow x_1 \neq 2$	1	2	2	$\{x_2 = 2 \rightarrow x_1 \neq 2\}$
$x_3 = 2 \rightarrow x_2 \neq 2$	1	1	2	$\{x_3 = 2 \rightarrow x_2 \neq 2\}$
$x_1 = 1 \rightarrow x_3 \neq 2$	1	1	1	$\{x_1 = 1 \rightarrow x_3 \neq 2\}$
$x_2 = 1 \rightarrow x_1 \neq 1$	2	1	1	$\{x_2 = 1 \rightarrow x_1 \neq 1\}$
$x_3 = 1 \rightarrow x_2 \neq 1$	2	2	1	$\{x_3 = 1 \rightarrow x_2 \neq 1\}$
$x_1 = 2 \rightarrow x_3 \neq 1$	2	2	2	$\{x_1 = 2 \rightarrow x_3 \neq 1\}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 1: Search states of a simple search algorithm.

The inference made by an ordered nogood is directed. The value of the conclusion variable  $y$  is ruled out based on the assignments of the antecedent variables  $x_j$ . In a traditional backtrack tree search  $x_j$  will therefore precede  $y$  in the variable ordering. This relative ordering condition will be denoted by  $x_j < y$ . One may wonder if  $\Gamma$  to ensure termination  $\Gamma$  it is not sufficient to eliminate the possibility for cyclic inferences. This can be done by requiring that the set of conditions  $x_j < y$  is acyclic. We hereby impose only a partial search order. Unfortunately  $\Gamma$  the answer is no. Consider the problem with three variables  $x_1$   $\Gamma$   $x_2$  and  $x_3$  each with values 1 and 2 and with constraints as depicted in figure 1. Table 1 shows the states of the search. The first column shows the nogood  $\tilde{\gamma}$  added at each step  $\Gamma$  the second column shows the current assignments and the third column  $\Gamma$  contains the current set of nogoods. Observe

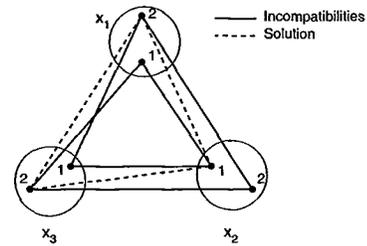


Figure 1: Example problem.

that the search process will cycle indefinitely  $\Gamma$  although  $x_1 = 2, x_2 = 1$  and  $x_3 = 2$  is a solution. The problem is that search information is deleted when nogoods are removed. As the example shows  $\Gamma$  this makes it possible for the search process to “chase its tail”.

The relative ordering conditions corresponding to the ordered nogoods are consequently not sufficient to ensure termination. PDB therefore imposes additional relative ordering conditions  $\Gamma$  called *safety conditions*. The idea is to retain some of the ordering conditions of the nogoods that have been removed. The order selected for new nogoods is now required to respect both types of ordering conditions. That is  $\Gamma$  the resulting set of conditions may not be cyclic.

In the example above  $\Gamma$  when  $x_2 = 2 \rightarrow x_1 \neq 2$  is deleted  $\Gamma$  PDB imposes the safety condition  $x_2 < x_1$ . Now we can no longer add  $x_1 = 1 \rightarrow x_3 \neq 2$  as it would cause a cycle. Instead  $\Gamma$  we rewrite it as  $x_3 = 2 \rightarrow x_1 \neq 1$ . This now prompts us to change the value of  $x_1$  to 2  $\Gamma$  thereby obtaining a solution.

### General Partial Order Backtracking

While DB imposes a total variable order  $\Gamma$  only a partial variable order is required by PDB. Nevertheless  $\Gamma$  PDB is not a generalization of DB. We illustrate this point with an example. Consider a problem where each variable has three values. Suppose  $\Gamma$  as illustrated on top in figure 2  $\Gamma$  that the values (shown as  $\bullet$ ) for  $x_1 \Gamma x_2$  and  $x_3$

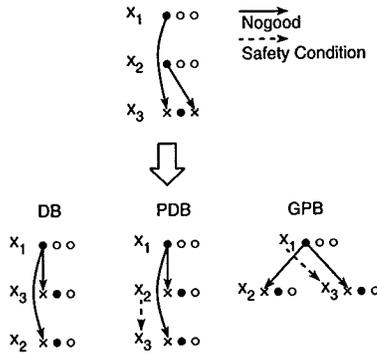


Figure 2: Backtracking with DB  $\Gamma$  PDB and GPB.

are not ruled out (shown as  $\times$ ) by any nogood. Suppose further that  $x_1$  and  $x_2$  rule out all values of a fourth variable  $x_4$  (not shown). At this point  $\Gamma$  we need to backtrack. So we use inference rule (2) to generate a new nogood involving  $x_1$  and  $x_2$ . Let us select  $x_2$  as conclusion and remove the nogood based on it. The first value of  $x_2$  is now eliminated. As depicted on the bottom of figure 2  $\Gamma$  PDB will hereby be forced to change the relative order of  $x_2$  and  $x_3 \Gamma$  whereas PDB has no choice but to keep the same order. In terms of flexibility both algorithms are therefore not comparable. However  $\Gamma$  it turns out that the safety condition  $x_2 < x_3$  can be relaxed. Below we describe a new algorithm called *general partial order backtracking* or GPB that exploits this fact. As illustrated on the bottom of figure 2  $\Gamma$  in this case GPB only requires that  $x_1 < x_3$ . Observe that for GPB both the order of DB and the one of PDB are admissible.

PDB specifies explicitly how the set of safety conditions changes with the addition of each nogood. However  $\Gamma$  it is not needed to do so explicitly. We can abstract from the details on how safety conditions actually change and instead describe general conditions  $\Gamma$  called *transition conditions*  $\Gamma$  that need to be respected. We will do so in the following way. By transitivity  $\Gamma$  a set of safety conditions  $S$  implies a set of ordering conditions that will be denoted by  $<_S$ . Instead of specifying which safety conditions  $<$  have to be present  $\Gamma$  we specify which ordering conditions  $<_S$  have to be verified. We

then do not need to concern ourselves with how safety conditions actually imply these ordering conditions.

We consider the addition of a single nogood. Let  $S$  and  $S'$  be the set of safety conditions  $\Gamma$  before and after the addition of this nogood. Similarly  $\Gamma$  let  $\Gamma$  and  $\Gamma'$  be the set of ordered nogoods before and after the nogood is added. Let the conclusion variable of the added nogood be  $y$  and let  $Z = \{z \mid y <_S z\}$ . The transition conditions we require are:

1. only ordering conditions of the type  $q <_S z$  with  $z \in Z$  may disappear  $\Gamma$
2. for every variable  $x$  for which  $x <_{S'} y \Gamma$  we have for all  $z \in Z \Gamma x <_{S'} z$  and
3. for every ordered nogood in  $\Gamma'$  with antecedent variables  $x_j$  and conclusion variable  $y \Gamma$  we have  $x_j <_{S'} y$ .

Aside from the generalization of the formulation  $\Gamma$  the main difference with respect to PDB is that we no longer require that  $y <_{S'} z$  for all  $z \in Z$ .

GPB has a main search loop and a backtrack procedure similar to PDB. The difference between the two resides in the conditions imposed on  $S$ . We use  $X$  to denote a complete set of assignments.

#### Algorithm GPB

Until  $X$  is a solution or the empty nogood is derived;  
 select a nogood  $\gamma$  corresponding to a constraint violation  $\Gamma$   
 Backtrack  $\gamma \Gamma$   
 change  $X$  to be acceptable for  $\Gamma$ .

#### Backtrack $\gamma$

Select a conclusion variable  $y$  of  $\gamma$  respecting  $S$ .  
 Add  $\bar{y}$  to  $\Gamma$  and remove from  $\Gamma$  all nogoods which have  $y$  as antecedent variable.  
 Modify  $S$  so that the transition conditions are satisfied.  
 If the live domain<sup>2</sup> of  $y$  is empty  
 infer a new nogood  $\rho$  and  
 Backtrack  $\rho$ .

In the next section we prove that the properties of GDB concerning termination and systematicity still hold for GPB. In the subsequent section  $\Gamma$  we demonstrate that GPB is a generalization of both DB and PDB.

### Properties of GPB

GPB goes through a sequence of states  $\Gamma$  called *partial order states*  $\Gamma$  each of which is described by a set of safety conditions  $S$  and a set of nogoods  $\Gamma$ . The transition between two subsequent states is performed by a top level call to the procedure Backtrack  $\Gamma$  including recursive calls if any. A state described by a total order and a complete set of assignments will be called a *total order state*. A total order state is an *instance* of a partial order state if its total order respects the safety conditions  $S$  and if its set of assignments is acceptable for

<sup>2</sup>The live domain of a variable is the set of values of its domain that is not ruled out by a conclusion of a nogood.

the set of nogoods  $\Gamma$  of the partial order state. We can now present the main result of this section:

**Theorem 1** *For every sequence of partial order states  $\sigma^k$ , there exists a sequence of total order states  $\sigma_j^k$  so that  $\sigma_j^k$  is an instance of  $\sigma^k$  and so that the sequence  $\sigma_j^k$  is visited during the execution of a systematic backtrack algorithm.*

Note that this is a stronger property than what was previously shown for GDB in (Ginsberg & McAllester 1994). In fact their result is a consequence of this theorem and is presented in corollary 1.

We first present three lemmas that allow us to prove theorem 1. In the lemmas we will consider the addition of a single nogood. That is recursive calls of the procedure Backtrack are treated individually. The first lemma is adapted from (Ginsberg & McAllester 1994) lemma 5.5) the two remaining ones appear to be new.

Let  $\alpha$  be the situation before  $\Gamma$  and  $\beta$  the situation after the addition of a single nogood. Take any total order that respects the ordering conditions in  $\beta$ . In this order let  $s$  be the last variable before any variable in  $Z \cup \{y\}$ . Let  $S_1$  be the set of variables up to and including  $s$  and let  $S_2$  be the remaining variables. Furthermore let  $S_1^\beta$  be the ordering of  $S_1$  according to the chosen order in  $\beta$  and  $S_2^\alpha$  an ordering of  $S_2$  that respects the ordering conditions in  $\alpha$ . The first two lemmas allow us to construct an order in  $\alpha$  for any order in  $\beta$ . In particular lemma 2 shows that this order can be constructed as depicted in figure 3.

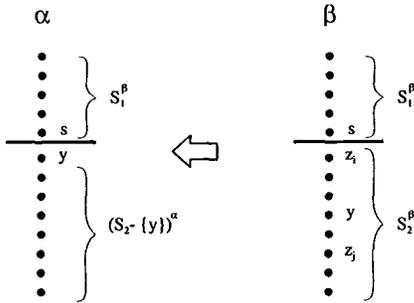


Figure 3: Orders before and after backtracking

**Lemma 1** *The order  $S_1^\beta; S_2^\alpha$  respects the ordering conditions in  $\alpha$ .*

**Proof:** Transition condition 1 ensures that only ordering conditions of the type  $q <_S z$  with  $z \in Z$  disappear. Since by construction  $Z \subseteq S_2$  we know that only ordering conditions of the type  $q <_S s_2$  with  $s_2 \in S_2$  are removed. Based on this we can verify that all ordering conditions in  $\alpha$  are satisfied by  $S_1^\beta; S_2^\alpha$ . Let us distinguish 4 types of ordering conditions:

1.  $s'_1 <_S s''_1$  with  $s'_1 \in S_1$  and  $s''_1 \in S_1$ . No ordering conditions of the type  $q <_S s_1$  with  $s_1 \in S_1$  have been removed between  $\alpha$  and  $\beta$ . So the ordering conditions

between elements in  $S_1$  are the same in  $\alpha$  as in  $\beta$ . They are therefore satisfied by  $S_1^\beta$ .

2.  $s'_2 <_S s''_2$  with  $s'_2 \in S_2$  and  $s''_2 \in S_2$ . For the elements of  $S_2$  an order  $S_2^\alpha$  is used. This order respects the ordering conditions of  $\alpha$  by definition.
3.  $s_1 <_S s_2$  with  $s_1 \in S_1$  and  $s_2 \in S_2$ . In  $S_1^\beta; S_2^\alpha$  the elements  $s_1$  precede the elements  $s_2$  by construction. All ordering conditions of the type  $s_1 <_S s_2$  are therefore satisfied by the proposed order.
4.  $s_2 <_S s_1$  with  $s_1 \in S_1$  and  $s_2 \in S_2$ . This type of ordering conditions is not present in  $\beta$  and was not removed when going from  $\alpha$  to  $\beta$ . This means that it could not be present in  $\alpha$ .

□

**Lemma 2** *The order  $S_1^\beta; y; (S_2 - \{y\})^\alpha$  respects the ordering conditions in  $\alpha$ .*

**Proof:** By lemma 1 we know that  $S_1^\beta; S_2^\alpha$  respects the ordering conditions in  $\alpha$ . So it suffices to show that  $y$  can be taken as first element in  $S_2^\alpha$ . We prove this by contradiction. Suppose  $y$  could not be taken as first element in  $S_2^\alpha$ . This would mean that there exists an ordering condition of the type  $x <_S y$  with  $x \in S_2 - \{y\}$ . By transition condition 1 this ordering condition could not have been removed when going from  $\alpha$  to  $\beta$ . Now by transition condition 2 in  $\beta$  we will also have  $x <_S z$  with  $z \in Z$ . But this would mean that  $x$  precedes both  $y$  and  $z$ . If this were the case  $x$  would have been part of  $S_1$  which is a contradiction. □

Theorem 1 and its proof refer to a systematic backtrack algorithm. For this purpose let us consider a backtrack search algorithm with dynamic variable reordering. The elementary step of the algorithm is to process a single nogood. Consider the addition of a new nogood with conclusion  $y$ . First all nogoods are removed that have  $y$  or subsequent variable as antecedent variable. Then the search tree is further explored to find a next set of assignments. If the live domain of  $y$  is not empty a complete set of assignments that satisfies all nogoods will be found. Otherwise the partial set of assignments up to and excluding  $y$  is used. Lemma 3 shows that this algorithm is systematic in the sense that any total set of assignments can at most be visited once. Furthermore when a value of a variable  $y$  is ruled out by a nogood we can choose to instantiate any future variable next<sup>3</sup>.

As before let  $\alpha$  and  $\beta$  be the situation before and after the addition of a single nogood. In lemma 3 and in the proof of theorem 1 assignments will be denoted as follows.  $X_1^\beta$  and  $X_2^\beta$  are the assignments in  $\beta$  for the variables in  $S_1$  and  $S_2$  respectively and  $(X_2 - \{y = v\})^\alpha$  are the assignments for the variables in  $S_2 - \{y\}$  in  $\alpha$ .

**Lemma 3** *The transition from the partial set of assignments  $X_1^\beta, y = v$  with search order  $S_1^\beta; y; (S_2 -$*

<sup>3</sup>This is in fact the modification that was applied to DBI in (Ginsberg 1993 page 32).

$\{y\}^\alpha$  to the partial set of assignments  $X_1^\beta$  with search order  $S_1^\beta; S_2^\beta$  is systematic.

**Proof:** The set of nogoods with conclusion variable in  $S_1$  has not been changed.  $X_1^\beta$  is therefore a consistent partial set of assignments for  $S_1$  in  $\alpha$ . Now the variables in  $S_1$  are ordered according to  $S_1^\beta$  and precede  $y$ . The addition of the nogood  $\tilde{\gamma}$  therefore rules out the portion  $X_1^\beta, y = v$  of the search space. The nogoods that will be dropped by the addition of  $\tilde{\gamma}$  have  $y = v$  or subsequent assignments as antecedent. Since the conclusion variable of these nogoods is in  $S_2 - \{y\}$  the portion they ruled out is therefore a strict subset of the portion  $X_1^\beta, y = v$  ruled out by  $\tilde{\gamma}$ . As a result the set of possible assignments is a strict subset of the one before the addition of the nogood.  $\square$

**Proof of Theorem 1:** Let  $\alpha$  and  $\beta$  now denote two subsequent partial order states of GPB. We first show that for any total order state  $\beta_j$  in  $\beta$  we can construct a total order state  $\alpha_i$  in  $\alpha$  so that  $\beta_j$  can be reached from  $\alpha_i$  by a systematic transition. This construction is illustrated in figure 4. Consider first the case where only

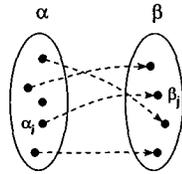


Figure 4: Transitions between instances of subsequent states.

one nogood is added in the transition between  $\alpha$  and  $\beta$ . By lemma 2 we know that for each order  $S_1^\beta; S_2^\beta$  in  $\beta$  we can construct the order  $S_1^\beta; y; (S_2 - \{y\})^\alpha$  in  $\alpha$ . Now by lemma 3 the transition between  $X_1^\beta, y = v$  ordered according to  $S_1^\beta; y; (S_2 - \{y\})^\alpha$  and  $X_1^\beta, X_2^\beta$  ordered according to  $S_1^\beta; S_2^\beta$  is systematic. This means that for  $\alpha_i$  we can take as assignments  $X_1^\beta, y = v, (X_2 - \{y = v\})^\alpha$  and as order  $S_1^\beta; y; (S_2 - \{y\})^\alpha$ . If more than one nogood is added between  $\alpha$  and  $\beta$  this reasoning can be applied recursively.

We can now repeat this constructive argument for any sequence of partial order states  $\sigma^k$  visited by GPB. By doing so we obtain sequence of total order states  $\sigma_j^k$  which are visited during the execution of a systematic backtrack algorithm (see figure 5).  $\square$

**Corollary 1** GPB terminates and visits at most  $\prod_{i=1}^n d_i$  states, where  $d_i$  is the domain size of variable  $x_i$ .

Let us illustrate the level of systematicity implied by theorem 1. Consider for example the problem of finding all solutions to a given CSP. In this case the algorithm needs to be modified in the following way. Instead of

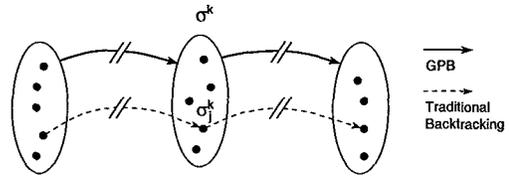


Figure 5: Sequence of instances visited by a systematic backtrack algorithm.

aborting the search when a solution is found the solution is stored and a new nogood is generated that rules out exactly this set of assignments. Now by theorem 1 this nogood will also be encountered in the execution of an underlying traditional backtrack algorithm. Since this algorithm is systematic it will not revisit this set of assignments. Hence it is not possible to encounter this nogood more than once. As a result GPB will enumerate every solution to the CSP exactly once.

Nevertheless GPB can revisit the same set of assignments. So like PDB it is not systematic in the traditional sense. To see this consider the problem with three variables  $x_1, x_2$  and  $x_3$  each with values 1 and 2. Table 2 shows a possible sequence of states in which the same set of assignments is revisited twice.

$\tilde{\gamma}$	$x_1$	$x_2$	$x_3$	$S$	$\Gamma$
	1	1	1		
$x_2 = 1 \rightarrow x_3 \neq 1$	2	1	2		$\{x_2 = 1 \rightarrow x_3 \neq 1\}$
$x_1 = 2 \rightarrow x_2 \neq 1$	2	2	2	$\{x_1 < x_3\}$	$\{x_1 = 2 \rightarrow x_2 \neq 1\}$
$x_1 \neq 2$	1	1	1		$\{x_1 \neq 2\}$

Table 2: GPB is not systematic in the traditional sense.

However we can make it systematic. All we need to do is to restrict<sup>4</sup> GPB to change only assignments in  $Z$ . Let us call this algorithm GPB'.

**Theorem 2** GPB' is systematic.

**Proof:** Consider the proof of theorem 1. Now with GPB' only the assignments in  $Z$  are allowed to be modified. The assignments in  $S_1$  will therefore remain unchanged. As a result the assignments constructed for the proof of theorem 1 will be the ones which are actually visited by GPB'. Since the traditional backtrack algorithm visiting these assignments is systematic GPB' is systematic as well.  $\square$

GPB is additive on disjoint subproblems for the same reason as PDB is. The variables appearing in the same nogood are connected. This is the case for nogoods that are generated by constraint violations and is inductively

<sup>4</sup>The difference between POB and PDB is that in POB only the value of the conclusion variable is changed. POB therefore satisfies the proposed restriction and is systematic.

true for the nogoods generated by inference rule (2). Nogoods are only removed when their antecedents are ruled out by the conclusion of some other nogood. We consequently have no interaction between nogoods of disjoint subproblems.

Observe that lemma 3 would allow us to also remove the nogoods with antecedent variables in  $Z$ . By doing so we would obtain a new algorithm for which theorem 1 holds. Note that an ordering relation can only exist between variables that have been connected by a nogood. This modified algorithm therefore still enjoys the additivity property.

## Instances of GPB

In this section we demonstrate that GPB is more general than DB and PDB in two ways. We first show that both algorithms are instances of GPB in the sense that they correspond with a specific choice for representing and manipulating  $S$ . Then we present an instance of GPB $\Gamma$  called GPB $\Gamma_t$  from which DB and GDB can be obtained by making heuristic choices.

DB<sup>5</sup> distinguishes instantiated variables  $i$  from uninstantiated variables  $u$ . All possible ordering conditions of the type  $i <_S u$  are therefore implied. Furthermore  $\Gamma$ DB assumes a total order on instantiated variables. This means that for every pair of variables  $i_k$  and  $i_l$   $\Gamma$ we have a condition of the type  $i_k <_S i_l$  if  $i_l$  follows  $i_k$  in the total order. When a new nogood is added with conclusion variable  $y$   $\Gamma$ DB effectively uninstantiates  $y$ . By doing so  $\Gamma$ all ordering conditions of the type  $y <_S z$  are removed  $\Gamma$ which is in accordance with transition condition 1. Now  $\Gamma$ let  $x$  be an instantiated variable that precedes  $y$  in the total order and let  $z$  be a variable that follows  $y$ . With DB  $\Gamma$  $x$  will still precede  $z$  after the reordering  $\Gamma$ whether  $z$  was an instantiated variable or a future variable. As required by transition condition 2  $\Gamma$ the ordering conditions of the type  $x <_{S'} z$  are therefore satisfied. Transition condition 3 is verified since all nogoods either respect the ordering conditions  $i_k <_S i_l$   $\Gamma$ or the ordering conditions  $i <_S u$ .

PDB manipulates safety conditions directly. When a nogood is added with conclusion variable  $y$   $\Gamma$ it removes safety conditions of the type  $q < z$  for each  $z \in Z$   $\Gamma$ where  $Z = \{z \mid y <_S z\}$ . By doing so  $\Gamma$ only ordering conditions of the type  $q' <_S z$  can disappear. Transition condition 1 is therefore verified. Then it adds the safety condition  $y < z$  for every  $z \in Z$ . This means that for each  $x$  for which  $x <_{S'} y$  we have  $y < z$  and hence  $x <_{S'} z$ . So that transition condition 2 is satisfied as well. Since PDB explicitly respects the set of ordering conditions corresponding to the ordered nogoods  $\Gamma$ transition condition 3 is satisfied.

<sup>5</sup>Since all nogoods with antecedents in  $Z$  may safely be removed  $\Gamma$ the same reasoning can be used to show that GDB (McAllester 1993) is an instance of GPB.

There are many other ways to guarantee that transition conditions are verified. Consider the two following instances:

**GPB $_S$**  The transition conditions can be coded explicitly by modifying PDB in the following way. Instead of adding the safety condition  $y < z$  for every  $z \in Z$   $\Gamma$ we can add the safety condition  $x < z$  for every  $x < y$ . This means that no relative ordering between  $y$  and  $z$  is imposed (see figure 2).

**GPB $_t$**  A different way of representing safety conditions is to associate with every nogood a set of variables called a *trail*. For a given nogood let the antecedent variables be  $x_i$   $\Gamma$ the conclusion variable be  $c$  and the trail variables be  $t_j$ . We now associate two sets of safety conditions  $\bar{S}$  and  $\underline{S}$  with the nogoods. In  $\bar{S}$  a nogood defines for each  $x_i$  the safety conditions;  $x_i < c$  and  $x_i < t_j$  for every  $t_j$ . In  $\underline{S}$  a nogood defines for each  $x_i$ ;  $x_i < c$  and for each  $t_j$ ;  $c < t_j$ .

Algorithm GPB $_t$  proceeds as follows. When a new nogood is added  $\Gamma$ a conclusion variable  $y$  is selected that respects  $\bar{S}$ . As before  $\Gamma$ the new set of nogoods  $\Gamma'$  is obtained by removing from  $\Gamma$  all nogoods which have  $y$  as antecedent variables. Now let  $Z = \{z \mid y <_S z\}$  as obtained from the set of nogoods  $\Gamma$ . Each  $z \in Z$  is removed from the trail of all nogoods. Finally  $\Gamma$ each  $z \in Z$  is added to the trail of those nogoods who either have  $y$  in their trail or as conclusion variable.

Observe that if  $Z$  was defined by  $\bar{S}$   $\Gamma$ we obtain algorithm GPB $_S$ . However  $\Gamma$ we are free to use  $\underline{S}$  to define  $Z$   $\Gamma$ since lemma 1 and lemma 2 will still hold. Indeed  $\Gamma$ referring to their proof  $\Gamma$ this eliminates some orders in  $\beta$  but not in  $\alpha$ .

GPB $_t$  generalizes both DB and PDB in the sense that these algorithms can be obtained by defining a specific heuristic. Let us see why this is the case. In GPB $_t$  we can choose to extend a consistent partial assignment. The corresponding variables are the instantiated variables of DB. Now when a new nogood is added we may effectively uninstantiate its conclusion  $y$  and place it in the set of future variables. Indeed  $\Gamma$ subsequent nogoods need only to respect  $\bar{S}$   $\Gamma$ not  $\underline{S}$ . This means that DB can be obtained by making specific heuristic choices in GPB $_t$ . Similarly  $\Gamma$ we can also make heuristic choices in GPB $_t$  so that it behaves as PDB. In particular  $\Gamma$ since  $\bar{S}$  is less restrictive than  $\underline{S}$   $\Gamma$ we can always order new nogoods as is done in PDB. If we do so  $\Gamma$  $\underline{S}$  of GPB $_t$  will be the same as  $S$  of GDB and  $Z$  will be the same as in GDB.

## Flexible Partial Order Backtracking

It turns out that the transition conditions of GPB can be further relaxed. By doing so we obtain an algorithm that enjoys even more freedom than GPB to explore the search space. We therefore call this algorithm *flexible partial order backtracking* or FPB. FPB is almost identical to GPB. The only difference is that transition condition 2 is now replaced by:

2. For every antecedent variable  $a_j$  in  $\bar{\gamma}$  we have  $a_j <_S z$  for all  $z \in Z$ .

The added flexibility of FPB comes at a price. Although we are able to show that the algorithm terminates it no longer enjoys the properties concerning systematicity.

To prove that FPB terminates we will show that monotonic progress is made according to some measure. The flexibility of FPB requires us to introduce a new measure called the *maximum space size tuple*. The *space size* of a variable  $x_j$  with respect to a set of nogoods  $\Gamma$  and a given order  $x_1; \dots; x_n$  is defined to be  $\sigma_j = \prod_{i \leq j} D_k(\Gamma, i)$  where  $D_k(\Gamma, i)$  is the live domain of  $x_k$  with respect to the nogoods in  $\Gamma$  whose antecedent variables  $x_i$  have  $i \leq j$ . The *space size tuple* with respect to a given order and set of nogoods  $\Gamma$  is the tuple  $(\sigma_1, \dots, \sigma_n)$ . The *maximum space size tuple* is the tuple which is lexicographically the largest over all admissible orderings.

As before let  $\alpha$  and  $\beta$  be the situations before and after a nogood  $\bar{\gamma}$  is added. Take any total order that respects the ordering conditions in  $\beta$ . Now in this order let  $a$  be the last antecedent variable of  $\bar{\gamma}$ .

**Lemma 4** *For any order in  $\beta$  there exists an order in  $\alpha$  that agrees with it up to and including  $a$ .*

**Proof:** Since transition condition 1 still holds lemma 1 applies. Furthermore we know that  $a \in S_1$  since  $a <_S y$  and  $a <_S z$ . Hence the order  $S_1^\beta; S_2^\alpha$  respects the ordering conditions in  $\alpha$  and agrees with the order  $S_1^\beta; S_2^\beta$  up to and including  $a$ .  $\square$

**Theorem 3** *FPB terminates.*

**Proof:** To prove termination we show that the maximum space size tuple decreases with each backtrack. No nogoods with antecedent variables before or including  $a$  were deleted. Ignoring the effect of  $\bar{\gamma}$  on the space size tuples by lemma 4 we would have that for every space size tuple in  $\beta$  there exists a space size tuple in  $\alpha$  whose entries agree up to and including  $\sigma_a$ . But now we added a nogood  $\bar{\gamma}$  whose conclusion variable is  $y$  and whose last antecedent variable is  $a$ . This will reduce  $D_y(\Gamma, a)$  by one and hence reduce  $\sigma_a$ . The maximum space size tuple of  $\beta$  therefore has to be smaller than the maximum space size tuple of  $\alpha$ .  $\square$

As shown in theorem 3 the algorithm terminates. However it is not systematic in that a set of assignments can be visited more than once. This is true even if we restrict FPB to change only assignments in  $Z$ . In fact if we use the same modification as with GPB to find all solutions FPB may enumerate solutions more than once.

## Summary and Future Work

The main goal of this paper is to show that polynomial space backtrackers can yet be made more flexible. We have presented an algorithm called GPB that generalizes both PDB and DB. We have proven that GPB enjoys the same properties as PDB concerning termination and

systematicity. Then we presented an algorithm called FPB which is even more flexible than GPB. However in this case the extra flexibility came at a price. Although we showed that FPB terminates the algorithm can visit more states than GPB.

The next step is to investigate how heuristics can translate this added flexibility into increased efficiency. On the one hand the performance of existing heuristics may improve. For example we would expect that using GPB we can adhere to the local gradient heuristic of GSAT more often than with PDB (Ginsberg & McAllester 1994). On the other hand new heuristics will need to be developed. For example it is not clear how to best order new nogoods or how the next constraint violation should be selected.

The overhead of GPB is similar to the one of PDB. However to date little has been done to devise efficient representations and algorithms for maintaining nogoods and ordering conditions during the search. In our opinion this technical aspect also deserves some attention.

## Acknowledgments

I would like to thank David McAllester for pointing out that PDB can be viewed as lifted classical backtracking. Thanks also to Gilles Trombettoni for his constructive comments on a draft of this paper.

This research work is supported through the ERCIM Fellowship Programme. It was performed in part at IIIA-CSIC in Bellaterra Spain sponsored by the European Commission and in part at LIA-EPFL in Lausanne Switzerland sponsored by the Swiss National Science Foundation under project number 21-50237.97.

## References

- Baker A. 1994. The hazards of fancy backtracking. In *AAAI'94: Proceedings of the Twelfth National Conference on Artificial Intelligence* 288-293.
- Ginsberg M. and McAllester D. 1994. GSAT and dynamic backtracking. In Doyle J.; Sandewall E.; and Torasso P. eds. *KR'94: Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning* 226-237.
- Ginsberg M. 1993. Dynamic backtracking. *Journal of Artificial Intelligence Research* 1:25-46.
- McAllester D. 1993. Partial order backtracking. Research Note Artificial Intelligence Laboratory MIT. <ftp://ftp.ai.mit.edu/people/dam/dynamic.ps>.
- Prosser P. 1993. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence* 9(3):268-299.
- Stallman R. and Sussman G. 1977. Forward reasoning and dependency directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence* 9:135-196.