k-relevant explanations for constraint programming

Samir Ouis[1], Narendra Jussien[1], and Patrice Boizumault[2]

[1] École des Mines de Nantes
4, rue Alfred Kastler – BP 20722
F-44307 Nantes Cedex 3 – France
{souis,jussien}@emn.fr
[2] GREYC, CNRS UMR 6072
Université de Caen, Campus 2,
F-14032 Caen Cedex – France
boizu@info.unicaen.fr

Abstract. This paper presents a set of tools based on explanations for constraint programming. These tools exploit k-relevant explanations which enable us to use several explanations, which can sometimes leads to better diagnosis. k-relevant explanations are introduced and used to provide: diagnosis tools (state analysis, contradiction analysis, constraint impact analysis), interaction tools (dynamic constraint addition/retraction simulation), as well as improved search techniques.

1 Introduction

Constraint programming has been proved extremely successful for modelling and solving combinatorial problems appearing in fields such as scheduling, resource allocation or design. Several languages and systems such as CHIP [1], CHOCO [2], GNUPROLOG [3], ILOG SOLVER [4] have been developed and widely spread. But these systems are helpless when the constraints network to solve has no solution. Indeed, only a `no solution` message is sent to the user who is left alone to find : why the problem has no solution; which constraint to relax in order to restore the coherence; etc.

These questions yield two different problems: *explaining* inconsistency and *restoring* consistency. Several theoretical answers have been provided to address those questions: QUICKXPLAIN [5] computes conflict-sets for configuration problems, [6] and [7] introduce tools to dynamically remove constraints, PALM [8] uses conflict-sets to address those issues and defines new search algorithms, [9] introduces constraint-specific tools for providing user-friendly solutions to constraint problems, [10] generates tree-like explanations and combines them with ordering heuristics and selection strategies to obtain better explanations according to a well-defined criterion, etc.

In this paper, we advocate for the use of *k-relevant* explanations [11]. The idea is to record bounded sets of explanations [12] (rather than a single one) based on relevance (how *far* is the validity of the explanation considering the current situation – this is evaluated with k). k-relevant explanations, by providing a relevance-based long term memory for explanations, are used to design interaction-based tools, diagnosis tools as well as improved search techniques.

This paper is organized as follows: we recall the definition and generation of conflict-sets and explanations within constraint programming in Section 2. Then, we introduce k-relevance-bounded explanations (Section 3) and give an illustrative example (Section 4). Before illustrating the use of k-relevant explanations in our tools (Section 5) we discuss the different uses of our explanations. Finally, we give a short overview of the implementation.

2 Conflict-sets and explanations for constraint programming

A *Constraint Satisfaction Problem* (CSP) is defined by a set of variables $V = \{v_1, v_2, \ldots, v_n\}$, their respective value domains $D_1, D_2, \ldots, D_n$ and a set of constraints $C = \{c_1, c_2, \ldots, c_m\}$. A solution of the CSP is an assignment of values to all the variables such that all constraints in C are satisfied. We denote by $\mathbf{sol}(V, C)$ the set of solutions of the CSP (V, C).

In the following, we consider variables domains as unary constraints. Moreover, the classical enumeration mechanism that is used to explore the search space is handled as a series of constraints additions (value assignments) and retractions (backtracks). Those particular constraints are called *decision constraints*.

2.1 Definitions

Let us consider a constraints system whose current state (*i.e.* the original constraints and the set of decisions made so far) is contradictory. A **conflict-set** (*a.k.a.* **nogood** [13]) is a subset of the current set of constraints of the problem that, left alone, leads to a contradiction (no feasible solution contains a nogood). A conflict-set can be partitioned into two parts[3]: a subset of the original set of constraints ($C' \subset C$ in Equation 1) and a subset of decision constraints introduced so far in the search (here $dc_1, \ldots, dc_k$).

$$\mathbf{sol}\,(V, (C' \wedge dc_1 \wedge ... \wedge dc_k)) = \emptyset \tag{1}$$

An operational viewpoint of conflict-sets can be made explicit by rewriting Equation 1 the following way:

$$C' \wedge \left(\bigwedge_{i \in [1..k] \setminus j} dc_i \right) \rightarrow \neg dc_j \tag{2}$$

[3] Notice that some special cases may arise. If $k < 1$, the considered problem is proved to be over-constrained; some constraints need to be removed. If $C' = \emptyset$, the set of decisions that have been made so far is itself contradictory. This can happen only if no propagation is done after a decision has been made.

Let us consider $dc_j : v_j = a$ in the previous formula. Equation 2 leads to the following result ($s(v)$ is the value of variable v in the solution s):

$$\forall s \in \mathbf{sol}\left(V, C' \wedge \left(\bigwedge_{i \in [1..k] \setminus j} dc_i\right)\right), s(v_j) \neq a \tag{3}$$

The left hand side of the implication in Equation 2 is called an **eliminating explanation** (explanation for short) because it justifies the removal of value a from the domain $d(v_j)$ of variable v_j. It is noted: $\texttt{expl}(v_j \neq a)$.

Explanations can be combined to provide new ones. Let us suppose that $dc_1 \vee \ldots \vee dc_j$ is the set of all possible choices for a given decision (set of possible values, set of possible sequences, etc.). If a set of explanations $C'_1 \rightarrow \neg dc_1$, ..., $C'_j \rightarrow \neg dc_j$ exists, a new conflict-set can be derived: $C'_1 \wedge \ldots \wedge C'_j$. This new conflict-set provides more information than each of the old ones.

For example, a conflict-set can be computed from the empty domain of a variable v (using explanations for each of the removed values):

$$\bigwedge_{a \in d(v)} \texttt{expl}(v \neq a) \tag{4}$$

As we can see, explanations are the key concept to compute conflict-sets. We therefore will focus on explanations in the remaining of this paper.

2.2 Storing explanations: k-relevance-bounded learning

There generally exists several explanations for the removal of a given value. Several different approaches were introduced to handle that multiplicity. *Dependency Directed Backtracking*[14] records all encountered explanations. The major inconvenience of this approach is its exponential space complexity. Indeed, the number of recorded explanations increases in a monotonic way. Various algorithms only keep a *single* explanation: *Dynamic Backtracking* [15] and its improvements (*MAC-DBT* [16], *Generalized Dynamic Backtracking* [17], *Partial-order Dynamic Backtracking* [18]) and *Conflict-directed BackJumping* [19]). The idea, common in all these algorithms, is to forget (erase) explanations which are not valid any more considering the current set of decision constraints. Space complexity therefore remains polynomial while ensuring the completeness of the algorithms. Unfortunately, this idea is not really compatible with debugging: only one explanation is kept and past information is completely lost.

Instead of recording only one explanation, a seemingly interesting idea is to keep information as long as a given criterion is verified:

- Time-bounded criterion: explanations are forgotten after a given time. This criterion is similar to *tabu* list management in *tabu* search [20].
- Size-bounded criterion: [21] have used a criterion defined in [22]. This criterion keeps only the explanations with a size lower or equal to a given value n. This criterion limits the spatial complexity, but may forget really interesting conflict-sets.

– Relevance-bounded criterion: explanations are kept if they are not *too far* from the current set of decision constraints. This concept (called k-relevance) has been introduced in [11] and focus explanations/conflict-set management to what is important: relevance *w.r.t.* the current situation.

Time and size-bounded recording do have a controllable space complexity. This is also the case for k-relevance learning (*cf.* Section 3). As we shall see, our tools are meant for the debugging and the dynamic analysis of programs: the space occupation overhead (compared to recording-free techniques) is well worth it.

3 k-relevance-bounded explanations

While solving a constraint problem, the current state of calculus can be described with two sets of constraints: R **the set of relaxed constraints** (decisions which have been undone during search, constraints which have been explicitly relaxed by the user, etc.) and A the set of active constraints (the current constraint store). $\langle A, R\rangle$ is called a *configuration*. Following [11], we can now define a k-relevant explanation as:

Definition 1. *k-relevant explanation ([11])*
*Let $\langle A, R\rangle$ be a configuration. An explanation e is said to be **k-relevant** if it contains at most $k-1$ relaxed constraints,* i.e. $|e \cap R| < k$.

In k-relevance-bounded learning, only k-relevant explanations are kept during search. Hence, several different explanations may be kept for a given value removal. Thus $\texttt{expl}(v \neq a)$ will not contain any more a single explanation but the set of k-relevant explanations recorded for the removal of value a from the domain $d(v)$ of variable v.

3.1 Managing k-relevant explanations

Computing k-relevant explanations k-relevant explanations, as regular explanations [12] can be computed during propagation. However, some issues arise (see example 1).

> **Example 1 (Example for explanation computation) :**
> Let us consider two variables v_1 and v_2. Let us assume that value a for v_1 is only supported by value b from v_2 in constraint c. Let us finally assume that b is removed from v_2 (a set of explanations being: $\{\{c_1, c_2\}, \{c_1, c_3\}, \{c_4, c_5\}\}$). This removal needs to be propagated.
> But, which explanation one should choose to compute the explanation of the value removal $v_1 \neq a$? Do we have to consider all the possibilities $\{c, c_1, c_2\}$, $\{c, c_1, c_3\}$ or $\{c, c_4, c_5\}$? Only one ?

As values are removed only once, we can focus on one particular explanation: the one which actually performs the removal. The explanation computed at removal time is called the *main* one. Only that explanation will be used to compute forthcoming explanations[4]. Moreover, this explanation is exactly the one that would have been computed by a classical approach (see Section 2.1). It is however worth noticing that this particular explanation completely depends on the order in which constraints are introduced and handled.

> **Example 1 (followed) :**
> Let us suppose that the *main* explanation for the removal of value b from v_2 is $\{c_1, c_2\}$.
> Thus, the removal $v_1 \neq a$ will be justified by $\{c, c_1, c_2\}$.

Evolution of k-relevant explanations We need to maintain the relevance information attached to stored explanations upon constraint additions and retractions. In both ways, the relevance of explanations may vary. The idea is to keep track of these variations and to forget explanations as soon as they become irrelevant. We organize the contents of the k-relevant explanations according to the relevance of its explanations. More precisely, all k-relevant explanations for a given removal $expl(v \neq a)$ are partitioned into k subsets, *i.e.* $expl(v \neq a) = \cup_{i \in [0..k-1]} expl(v \neq a, i)$. An explanation $e \in expl(v \neq a, i)$ if $|e \cap R| = i$ with R the set of relaxed constraints.

> **Example 2 (Updating 2-relevant explanations) :**
> Let us consider the previous example (example 1). Let us assume that we have found other explanations for the removal $expl(v_1 \neq a)$. The 2-relevant explanations for this removal are : $expl(v_1 \neq a, 0) = \{\{c, c_1, c_2\}, \{c_2, c_4\}\}$, $expl(v_1 \neq a, 1) = \{\{c_1, c_3\}, \{c_2, c_3\}\}$.
> Notice that the constraint c_3 is relaxed. This is why the two explanations $\{\{c_1, c_3\}, \{c_2, c_3\}\}$ belong to the second second subset $expl(v_1 \neq a, 1)$.
> If we relax constraint c_1, explanation $\{c, c_1, c_2\}$ jumps from the $expl(v_1 \neq a, 0)$ subset to the $expl(v_1 \neq a, 1)$ subset while the explanation $\{c_1, c_3\}$ will be forgotten and removed from the $expl(v_1 \neq a, 1)$ subset when $k = 2$. After the constraint removal, the 2-relevant explanations become : $expl(v_1 \neq a, 0) = \{\{c_2, c_4\}\}$, $expl(v_1 \neq a, 1) = \{\{c, c_1, c_2\}, \{c_2, c_3\}\}$.
> Conversely, if we add the relaxed constraint c_3, the explanation $\{c_1, c_3\}$ and $\{c_2, c_3\}$ become valid and the 2-relevant explanations become : $expl(v_1 \neq a, 0) = \{\{c, c_1, c_2\}, \{c_2, c_4\}, \{c_1, c_3\}, \{c_2, c_3\}\}$, $expl(v_1 \neq a, 1) = \emptyset$.

Computing conflict-sets The same dilemma that we encountered when computing explanations appears when computing conflict-sets. Indeed, when a contradiction is identified (a domain of a variable becomes empty), we saw that Equation 4 computes a conflict-set. However, there may exist several explanations for each considered value removal. Contrarily to the explanation computation process, we chose here to provide all possible explanations (limiting ourselves

[4] This implies that we will never willingly compute the complete set of k-relevant explanations for a given value removal. We only keep track of *encountered* explanations.

to valid explanations *i.e.* $expl(v \neq a, 0))$ for all $a \in d(v)$. The resulting number of valid conflict-sets is:

$$\prod_{a \in d(v)} |expl(v \neq a, 0)| \tag{5}$$

Example 3 (The possible conflict-sets) :
Let us consider the previous example. Let us assume that the two values a and b for the domain of v_1 are removed with the following explanations : $expl(v_1 \neq a, 0) = \{\{c, c_1, c_2\}, \{c_2, c_4\}\}$ and $expl(v_1 \neq b, 0) = \{\{c_1, c_5\}, \{c_2, c_5\}\}$.
We can obtain the 4 following conflict-sets :

- $\{c, c_1, c_2\} \cup \{c_1, c_5\}$
- $\{c, c_1, c_2\} \cup \{c_2, c_5\}$
- $\{c_2, c_4\} \cup \{c_1, c_5\}$
- $\{c_2, c_4\} \cup \{c_2, c_5\}$

We notice that the main explanation is the one recorded among the valid one. For example, in our case, $\{c, c_1, c_2\}$ is the main explanation for the removal $expl(v_1 \neq a)$ and $\{c_1, c_5\}$ is the main explanation for the removal $expl(v_1 \neq b)$.

3.2 Complexity issues

To compute the complexity of our approach, let us consider a CSP defined upon n discrete variables with maximum domain size d upon which are posted e constraints. If we only keep a single explanation per value removal, there will be at most $n \times d$ explanations of maximal size $e + n$ *i.e.* all the constraints from the problem (e) and the decision constraints (n). Thus the complexity of the classical approach is $O((e + n) \times n \times d)$.

However, as far as the k-relevance approach is concerned, an explanation can contain up to $k-1$ relaxed constraints, the maximal size of an explanation being $n+e+k-1$. The maximum number of explanations for a given value removal is bounded by the maximum number of non included subsets in a set. The worst case is: $\binom{e+n+k-1}{(e+n+k-1)/2}$ subsets of size $(e + n + k - 1)/2$.

Therefore, the spatial complexity for storing k-relevance explanations is in:

$$O\left(n \times d \times \binom{e+n+k-1}{(e+n+k-1)/2} \times (e+n+k-1)/2\right)$$

3.3 Discussion

- **Difference between classical approaches and 1-relevance**
  All classical approaches (*eg. Dynamic Backtracking* or *MAC-DBT*) forget explanations as they become invalid. A 1-relevant learning technique will obviously proceed the same way. However, it differs from classical approaches

by the number of recorded explanations by removal. Indeed, during resolution, one may come across an explanation for an already performed removal. Instead of not taking it into account, 1-relevance will keep that secondary information[5].
Furthermore, all classical approaches take into account only one conflict-set. This conflict-set is computed following to Equation 4. Our approach may deal with a *set* of conflict-sets (see Equation 5). Nevertheless that particular explanation management has a computational and spatial cost.

– **How to compute the best conflict-set?**
The most interesting are those which are minimal regarding inclusion. The minimal one can be obtained by computing the covering of all the valid explanations. Unfortunately, computing such set is exponentially costly. The simplest conflict-set consists in computing the union of all the valid explanations but thus precision is lost. We claim that a good compromise between both criteria (precision and ease of computation) is to select one explanation by removed value (*i.e.* the main one) of the failing variable then to union them. The main explanation can be the first one computed (as presented above) or, for example, the one chosen by a comparator which will take the user's preferences into account.

4 An example : the conference problem

To illustrate the use of the k-relevant explanations, we present the resolution of the conference problem [23]. From now on, we fill focus our study to 1-relevance.

4.1 Presentation of the problem

Michael, Peter and Alan are organizing a two-day seminar for writing a report on their work. In order to be efficient, Peter and Alan need to present their work to Michael and Michael needs to present his work to Alan and Peter (actually Peter and Alan work in the same lab). Those presentations are scheduled for a whole half-day each. Michael wants to know what Peter and Alan have done before presenting his own work. Moreover, Michael would prefer not to come the afternoon of the second day because he has got a very long ride home. Finally, Michael would really prefer not to present his work to Peter and Alan at the same time.

4.2 A constraint model for the conference problem

A constraint model for that problem is described as follows : let Ma, Mp, Am, Pm the variables representing four presentations (M and m are respectively for Michael as a speaker and as an auditor). Their domain will be $[1, 2, 3, 4]$ (1

[5] It will be used to compute conflict-sets. The *main* explanation will still be the only one used to compute subsequent explanations.

is for the morning of the first day and 4 for the afternoon of the second day). Several constraints are contained in the problem: implicit constraints regarding the organization of presentations and the constraints expressed by Michael.

The implicit constraints can be stated:

- A speaker cannot be an auditor in the same half-day. This constraint is modelled as: $c_1 : Ma \neq Am$, $c_2 : Mp \neq Pm$, $c_3 : Ma \neq Pm$ and $c_4 : Mp \neq Am$.
- No one can attend two presentations at the same time. This is modelled as $c_5 : Am \neq Pm$.

Michael constraints can be modelled:

- Michael wants to speak after Peter and Alan: $c_6 : Ma > Am$, $c_7 : Ma > Pm$, $c_8 : Mp > Am$ and $c_9 : Mp > Pm$.
- Michael does not want to come on the fourth half-day: $c_{10} : Ma \neq 4$, $c_{11} : Mp \neq 4$, $c_{12} : Am \neq 4$ and $c_{13} : Pm \neq 4$.
- Michael does not want to present to Peter and Alan at the same time: $c_{14} : Ma \neq Mp$.

4.3 Using classical approaches

Table 1 shows the resulting explanations for both approaches (classical and 1-relevant) after adding constraints from c_1 to c_6.

The column associated with the classical approach contains only a single explanation by removal, opposed to the 1-relevant column. The domain of the variable P_m is empty. We deduce that our problem is over-constrained. According to the Equation 4 of the Section 2, we obtain the conflict-set $\{c_1, c_2, c_3, c_4, c_5, c_6\}$.

4.4 Solving using 1-relevant explanations

Table 2 presents the 1-relevant explanations associated with every removal after we have removed the redundant explanations like $\{c_5, c_6\}$ for the removal $P_m \neq 4$. But as the second approach proposes several explanations, we can deduce several conflict-sets. In our case, we obtain two conflict-sets : $\{c_1, c_3, c_4, c_5, c_6\}$ and $\{c_1, c_4, c_5, c_6\}$.

The second conflict-set is more precise since it is included in the first one. There is a quite important difference between the conflict-set provided by the first approach which contains all the constraints that do not help the user and the conflict-sets provided by the 1-relevant approach.

5 Exploiting k-relevant explanations

k-relevance provides more interesting explanations and allows to obtain a better diagnosis. In this section, we present several concrete situations which the user is frequently confronted in the case of a failure. We show how k-relevant explanations allow a better understanding and fine analysis of failures, a quick simulation of constraint-based scenarios, etc.

Table 1. Domains after the introduction of constraints

Variable	Value	Explanation	1-relevance	present ?
P_m	1	$\{c_1, c_2, c_4, c_6\}$	$\{c_1, c_2, c_4, c_6\}, \{c_1, c_4, c_6\}$	no
P_m	2	$\{c_5, c_6\}$	$\{c_5, c_6\}$	no
P_m	3	$\{c_5, c_6\}$	$\{c_5, c_6\}$	no
P_m	4	$\{c_3\}$	$\{c_3\}, \{c_5\}, \{c_5, c_6\}$	no
A_m	1	$\emptyset$	$\emptyset$	**yes**
A_m	2	$\{c_4, c_6\}$	$\{c_4, c_6\}$	no
A_m	3	$\{c_4, c_6\}$	$\{c_4, c_6\}$	no
A_m	4	$\{c_2\}$	$\{c_2\}, \{c_4\}, \{c_4, c_6\}$	no
M_p	1	$\{c_4\}$	$\{c_4\}, \{c_5\}, \{c_6\}$	no
M_p	2	$\emptyset$	$\emptyset$	**yes**
M_p	3	$\{c_6\}$	$\{c_6\}$	no
M_p	4	$\{c_6\}$	$\{c_6\}$	no
M_a	1	$\{c_2\}$	$\{c_2\}, \{c_3\}$	no
M_a	2	$\emptyset$	$\emptyset$	**yes**
M_a	3	$\emptyset$	$\emptyset$	**yes**
M_a	4	$\emptyset$	$\emptyset$	**yes**

Table 2. Final set of explanations

Variable	Value	Explanation	1-relevance	present ?
P_m	1	$\{c_1, c_2, c_4, c_6\}$	$\{c_1, c_4, c_6\}$	no
P_m	2	$\{c_5, c_6\}$	$\{c_5, c_6\}$	no
P_m	3	$\{c_5, c_6\}$	$\{c_5, c_6\}$	no
P_m	4	$\{c_3\}$	$\{c_3\}, \{c_5\}$	no
A_m	1	$\emptyset$	$\emptyset$	**yes**
A_m	2	$\{c_4, c_6\}$	$\{c_4, c_6\}$	no
A_m	3	$\{c_4, c_6\}$	$\{c_4, c_6\}$	no
A_m	4	$\{c_2\}$	$\{c_2\}, \{c_4\}$	no
M_p	1	$\{c_4\}$	$\{c_4\}, \{c_5\}, \{c_6\}$	no
M_p	2	$\emptyset$	$\emptyset$	**yes**
M_p	3	$\{c_6\}$	$\{c_6\}$	no
M_p	4	$\{c_6\}$	$\{c_6\}$	no
M_a	1	$\{c_2\}$	$\{c_2\}, \{c_3\}$	no
M_a	2	$\emptyset$	$\emptyset$	**yes**
M_a	3	$\emptyset$	$\emptyset$	**yes**
M_a	4	$\emptyset$	$\emptyset$	**yes**

5.1 Diagnosis tools

k-relevant explanations as regular ones are obviously useable for diagnosis purposes.

Providing more precise explanations If we have two explanations for the same removal e_1 and e_2 such as $e_1 \subsetneq e_2$, then we know that the constraints which belong to $e_2 \setminus e_1$ are not responsible for the removal. We say that e_1 is more precise than e_2.

Consequently, constraints belonging to $e_2 \setminus e_1$ are not responsible for the incoherence if they do not appear in the other removals. k-relevance is, of course, not the panacea (see constraint c_6 which is not responsible for the removal $P_m \neq 4$ but intervenes in the removal $Pm \neq 1$ – it appears in the conflict-set). But, the multiplicity of explanations leads to more precise explanations and conflict-sets.

Analyzing the impact of a constraint An interesting feature when debugging is to know whether a given constraint belongs to a conflict-set or not. k-relevant explanations help answer that question.

Let us suppose that the cause of incoherence is the variable A_m (see table 3). As there is a failure (the domain of variable A_m is empty), the user wants to know if *constraint* c_5 *belongs to a conflict-set* by only referring to table 3. Based on the classical approach, the only conflict-set would be $\{c_2, c_3, c_4, c_6\}$. Indeed, The answer will be negative ($c_5 \notin \{c_2, c_3, c_4, c_6\}$). While our 1-relevant approach provides 8 conflict-sets and indicates that the constraint c_5 is strongly responsible for the incoherence (it removes 3 values out of 4 in A_m).

Table 3. New state of the variable A_m

Variable	Value	Explanation	1-relevance	not deleted ?
A_m	1	$\{c_3\}$	$\{c_3\}, \{c_5\}$	no
A_m	2	$\{c_4, c_6\}$	$\{c_4, c_6\}, \{c_5\}$	no
A_m	3	$\{c_4, c_6\}$	$\{c_4, c_6\}, \{c_5\}$	no
A_m	4	$\{c_2\}$	$\{c_2\}, \{c_4\}$	no

Providing error diagnosis Imagine now that after some relaxations, the user wants to know why variable M_p cannot take the value 1? The classical approach provides the explanation $\{c_6\}$. While the 1-relevant approach provides the set of explanations: $\{\{c_4\}, \{c_5\}, \{c_6\}\}$. We see that 1-relevance provides a richer diagnosis than the classical approach.

User interaction As we can see in our examples, explanations (and thus k-relevant explanations) are sets of low-level constraints. Only a specialist can understand and correctly interpret the provided information because those constraint are very far from the end-user's vision of the solved problem. We introduced in [24] a way of providing user-readable explanations by using a tree-based representation of the user's understanding of the solved problem.

5.2 Interaction-based tools

k-relevance allows the simulation of constraint addition/retraction with no computational costs.

Simulating constraint relaxation Determining if a given constraint belongs to a conflict-set or not may lead to spending a lot of time by relaxing each suspected constraint. For that reason, we propose a tool which allows to simulate a relaxation (without any propagation) only by updating the k-relevant explanations.

For example, let us suppose that the user suspects that constraint c_3 belongs to a conflict-set. The constraint-checking tool (see Section 5.1) confirms it. The relaxation of this constraint will put back all the values a such as $c_3 \in expl(A_m \neq a)$. According to table 3, the constraint c_3 is partly responsible for the removal $A_m \neq 1$. The classical approach would have put back the value 1 in the domain of A_m and launched the propagation phase. Unfortunately, the problem is always over-constrained because the removal $Am \neq 1$ is justified by the constraint c_5 and the domain of A_m becomes empty again.

1-relevant explanations allow us to know that the relaxation of the constraint c_3 will lead to an another failure due to the removal $A_m \neq 1$ which will be justified by another explanation: $\{c_5\}$. Thus, our tool is able to indicate to the user if a relaxation of a given constraint will lead to another *immediate* failure with no computational cost.

Simulating constraint addition To solve a dynamic problem, a simple execution from scratch is too expensive for every modification introduced by the user. Some tools allowing to incrementally solve the problem from the current solution do not allow to know if this modification (more precisely the addition of a previously relaxed constraint) will lead to a future immediate failure.

It is helpful to take advantage of the information accumulated during the resolution of the previous problem to avoid adding constraints leading to an *immediate* failure. For this reason, we propose a tool simulating the re-introduction of a relaxed constraint without any propagation. This tool informs the user if the addition of a relaxed constraint leads to a failure or not.

For example, let us suppose now that the user have removed the constraints $\{c_3, c_5\}$ to put back the value 1 in the domain of A_m and we have obtained the table 4. Later, the user wants to put back the relaxed constraint c_3. Unfortunately, the classical approach would have put back the constraint c_3 and

Table 4. New state of the variable A_m after relaxation of c_3 and c_5

Variable	Value	Explanation	1-relevance	2-relevance	not deleted ?
A_m	1			$\{c_3\}, \{c_5\}$	yes
A_m	2	$\{c_4, c_6\}$	$\{c_4, c_6\}, \{c_5\}$		no
A_m	3	$\{c_4, c_6\}$	$\{c_4, c_6\}, \{c_5\}$		no
A_m	4	$\{c_2\}$	$\{c_2\}, \{c_4\}$		no

naively launched the propagation phase leading to a contradiction. However, the k-relevance approach can simulate this constraint comeback by updating the 2-relevance explanations and verifying at the same time if a domain becomes empty or not. In our case, if we add c_3, the 2-relevant explanation $\{c_3\}$ becomes valid (*i.e.* 1-relevant) and it will remove the only remaining value 1 in the domain of A_m. Therefore, the k-relevance approach informs the user that the addition of constraint c_3 will lead to a contradiction.

For implementing such a tool, we must decrease the relevance of the k-relevant explanations which contain the relaxed constraint c. If we add constraint c, some of them can become valid (*i.e.* 1-relevant). This will imply the removal of some values. Thus, we can easily verify if any domain becomes empty (*i.e.* a contradiction is identified) when adding constraint c.

5.3 Improving explanation-based techniques

We started to experiment the use of k-relevant explanations within explanation-based search algorithms such as [16]. Our very first preliminary results seem to show that when k increases, the performance of k-relevance decreases. Moreover, for $k = 1$ and $k = 2$, we obtain the same temporal performances as *MAC-DBT*[16]. Precisely, for $k = 1$ and $k = 2$, the time lost to manage the k-relevant explanations is compensated by avoiding failures. For $k \geq 3$, k-relevance loses its advantages. More precisely, for such a k, we are losing time managing explanations which will seldom let us avoid future failures. Especially, we shall update explanations which will never be valid (*i.e.* 1-relevant).

Therefore, a good compromise between recording useful explanations and k-relevant explanation time management, would be to use $k = 1$ or $k = 2$. Unfortunately, we pay a small penalty of memory space. However, further experiments need to be done for a more in-depth analysis of the interest of k-relevant explanations for improving search algorithms.

6 Implementation

Our tools have been implemented in `choco`[6] [2] using the `PaLM` system [8]. `choco` is a complete constraint solver whereas `PaLM` is a `choco` library adding explanation-based capabilities to `choco`.

Implementing k-relevant explanations modifies the way constraints are added or retracted:

- **Adding a constraint**
  1. **Connecting** the added constraint to the constraint network.
  2. **Decreasing** the relevance of the k-relevant explanations which contain this added constraint. Among these k-relevant explanations, some of them can become valid (*i.e.* the explanations restricted to that single constraint) and allow to remove some values not yet removed.
  3. **Verifying** if no domain becomes empty (*i.e.* looking for a contradiction).
  4. **Re-propagating** if some values were removed without contradiction, those removals must be propagated in order to maintain a consistent state.

  This process is different from the process used in the classical approaches [6, 7] as we do not need to compute the past effects of the added constraint because when we retract a constraint, we do not forget the associated explanations to this constraint as the classical approaches do.
- **Retracting a constraint**
  1. **Disconnecting** the retracted constraint from the constraint network.
  2. **Increasing** the relevance of the k-relevant explanations and undoing the past effects of the constraint. Also, we forget explanations that become irrelevant.
  3. **Re-propagation** in order to have a consistent state.

  At the end of this process, the system is in a consistent state. It is exactly the state that would have been obtained if the retracted constraint would not have been introduced into the system.

7 Conclusion

In this paper, we have proposed the foundations of several interactive tools which are of great help for a user to solve contraint programming. We have shown the effectiveness of these k-relevance-based tools. k-relevance is a relevance-bounded recording technique for explanations.

[6] `choco` is an open source constraint engine developed as the kernel of the OCRE project. The OCRE project (its name standing for *Outil Contraintes pour la Recherche et l'Enseignement*, *i.e.* , *Constraint tool for Research and Education*) aims at building free Constraint Programming tools that anyone in the Constraint Programming and Constraint Reasoning community can use. For more information see www.choco-constraints.net.

We have shown the contribution of our k-relevance-based tools compared to classical approaches. k-relevance provides more precise explanations; gives some general information that is not accessible within a classical framework: k-relevance allows the simulating of constraint retraction/addition and so provides richer diagnosis tools.

Our current work includes designing algorithms which compute the *best* conflict-set (see Section 3.3) from all the explanations. For this, we plan to add user-based comparators [25] in order to provide automatic comparison of solutions. Also, we are trying to decrease the space required to manage the explanations.

References

1. Aggoun, A., Dincbas, M., Herold, A., Simonis, H., Van Hentenryck, P.: The CHIP System. Technical Report TR-LP-24, ECRC, Munich, Germany (1987)
2. Laburthe, F.: CHOCO: implementing a CP kernel. In: CP00 Post Conference Workshop on Techniques for Implementing Constraint programming Systems (TRICS), Singapore (2000)
3. Diaz, D., Codognet, P.: The GNU prolog system and its implementation. In: ACM Symposium on Applied Computing, Villa Olmo, Como, Italy (2000)
4. Ilog: Solver reference manual (2001)
5. Junker, U.: QUICKXPLAIN: Conflict detection for arbitrary constraint propagation algorithms. In: IJCAI'01 Workshop on Modelling and Solving problems with constraints, Seattle, WA, USA (2001)
6. Bessière, C.: Arc consistency in dynamic constraint satisfaction problems. In: Proceedings AAAI'91. (1991)
7. Debruyne, R.: Arc-consistency in dynamic CSPs is no more prohibitive. In: 8^{th} Conference on Tools with Artificial Intelligence (TAI'96), Toulouse, France (1996) 299–306
8. Jussien, N., Barichard, V.: The palm system: explanation-based constraint programming. In: Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000, Singapore (2000) 118–133
9. Squali, M.H., Freuder, E.C.: Inference-based constraint satisfaction supports explanation. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI '96). Volume 1. (1996) 318–324 Portland, Oregon.
10. Freuder, E.C., Likitvivatanavong, C., Wallace, R.J.: A case study in explanation and implication. In: In CP2000 Workshop on Analysis and Visualization of Constraint Programs and Solvers. (2000)
11. Bayardo Jr., R.J., Miranker, D.P.: A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In: AAAI'96. (1996)
12. Jussien, N.: e-constraints: explanation-based constraint programming. In: CP01 Workshop on User-Interaction in Constraint Satisfaction, Paphos, Cyprus (2001)
13. Schiex, T., Verfaillie, G.: Nogood Recording fot Static and Dynamic Constraint Satisfaction Problems. International Journal of Artificial Intelligence Tools **3** (1994) 187–207
14. Stallman, R.M., Sussman, G.J.: Forward reasoning and dependency directed backtracking in a system for computer-aided circuit analysis. Artificial Intelligence **9** (1977) 135–196

15. Ginsberg, M.: Dynamic backtracking. Journal of Artificial Intelligence Research **1** (1993) 25–46
16. Jussien, N., Debruyne, R., Boizumault, P.: Maintaining arc-consistency within dynamic backtracking. In: Principles and Practice of Constraint Programming (CP 2000). Number 1894 in Lecture Notes in Computer Science, Singapore, Springer-Verlag (2000) 249–261
17. Bliek, C.: Generalizing partial order and dynamic backtracking. In: Proceedings of AAAI. (1998)
18. Ginsberg, M., McAllester, D.: GSAT and dynamic backtracking. In Borning, A., ed.: Principles and Practice of Constraint Programming. Volume 874 of Lecture Notes in Computer Science., Springer (1994) (PPCP'94: Second International Workshop, Orcas Island, Seattle, USA).
19. Prosser, P.: MAC-CBJ: maintaining arc-consistency with conflict-directed backjumping. Research Report 95/177, Department of Computer Science – University of Strathclyde (1995)
20. Glover, F., Laguna, M.: Modern heuristic Techniques for Combinatorial Problems, chapter Tabu Search, C. Reeves. Blackwell Scientific Publishing (1993)
21. Schiex, T., Verfaillie, G.: Nogood recording for static and dynamic CSP. In: 5^{th} IEEE International Conference on Tools with Artificial Intelligence, Boston, MA. (1993) 48–55
22. Dechter, R.: Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. Artificial Intelligence **41** (1990) 273–312
23. Jussien, N., Boizumault, P.: Implementing constraint relaxation over finite domains using ATMS. In Jampel, M., Freuder, E., Maher, M., eds.: Over-Constrained Systems. Number 1106 in Lecture Notes in Computer Science, Springer-Verlag (1996) 265–280
24. Jussien, N., Ouis, S.: User-friendly explanations for constraint programming. In: ICLP'01 11th Workshop on Logic Programming Environments (WLPE'01), Paphos, Cyprus (2001)
25. Borning, A., Maher, M., Martindale, A., Wilson, M.: Constraint hierarchies and logic programming. In Levi, G., Martelli, M., eds.: ICLP'89: Proceedings 6th International Conference on Logic Programming, Lisbon, Portugal, MIT Press (1989) 149–164