COINS: a constraint-based interactive solving system

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 describes the COINS (COnstraint-based INteractive Solving) system: a conflict-based constraint solver. It helps understanding inconsistencies, simulates constraint additions and/or retractions (without any propagation), determines if a given constraint belongs to a conflict and provides diagnosis tools (*eg.* why variable v cannot take value val). COINS also uses user-friendly representation of conflicts and explanations.

1 Introduction

Constraint programming has been proved extremely successful for modelling and solving combinatorial search problems appearing in fields such as scheduling resource allocation and 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 constraint 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 conflicts for configuration problems, [6] and [7] introduce tools to dynamically remove constraints, PALM [8] uses conflicts to address those issues and define new search algorithms, etc.

User interaction requires user-friendly and interactive tools. In this paper, we advocate for the use of *k-relevant* explanations [9] to provide the COINS (COnstraint-based INteractive Solving) system.

COINS helps the user understand inconsistency, simulate constraint additions and/or retractions (without any propagation), determine if a given constraint belongs to a conflict and provide diagnosis tools (*eg.* why variable v cannot take value val). COINS is based upon the use of conflict sets (*a.k.a.* nogood [10]), explanations [11] and their user-friendly representation [12].

This paper is organized as follows: we review the definition and generation of conflicts 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 the COINS system (section 6) we present a natural way to provide user-friendly explanations (section 5). Finally, we give a short overview of our implementation.

2 Conflict 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*. This rather unusual consideration allow the easy generalization of the concepts that are presented in this paper to any kind of decision constraints (not only assignments but also precedence constraints between tasks for scheduling problems or splitting constraints in numeric CSP, etc.).

2.1 Definitions

Let us consider a constraints system whose current state (*i.e.* the original constraint and the set of decisions made so far) is contradictory. A **conflict set** (*a.k.a.* **nogood** [10]) is a subset of the current constraints system of the problem that, left alone, leads to a contradiction (no feasible solution contains a nogood). A conflict divides 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 $dc_1, \ldots, dc_k$).

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

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

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

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 $d(v)$ of variable v. It is noted: $expl(v \neq a)$.

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 can be derived: $C'_1 \wedge \ldots \wedge C'_j$. This new conflict provides more information than each of the old ones.

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

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

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*[13] records all encountered explanations. The major inconvenience of this approach is its exponential space complexity. Indeed, the number of recorded explanations increases in a monotonous way. Various algorithms only keep a *single* explanation: *Dynamic Backtracking* [14] and its improvements

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

(*MAC-DBT* [15], *Generalized Dynamic Backtracking* [16], *Partial-order Dynamic Backtracking* [17]) and *Conflict-directed BackJumping* [18]. The idea 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 are completely lost.

Instead of recording only one explanation, a more 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 criteria is similar to *tabu* list management in *tabu* search [19].
– Size-bounded criterion: [20] have used a criteria defined in [21]. This criteria keeps only the explanations with a size lower or equal to a given value n. This criteria limits the spatial complexity, but may forget really interesting nogoods.
– 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 [9] and focus explanations/conflict management to what is important: relevance *wrt* 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 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 that have been undone during search, constraints that 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 [9], we can now define a k-relevant explanation as:

Definition 1. *k-relevant explanation ([9])*
*Let $\langle A, R\rangle$ 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 $expl(v \neq a)$ 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 [11] 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 from v_1 is only supported by value b from v_2 in constraint c. Let 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 first one. The explanation computed at removal time is called the *main* one. That explanation will be used to compute forthcoming explanations. Moreover, this explanation is exactly the one that would have been computed by a classical approach (see section 2.1).

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 the 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.* $expl(v \neq a) = \cup_{i\in[0..k-1]} expl(v \neq a, i)$. Subset $expl(v \neq a, i)$ contains the explanations having i relaxed constraints.

Example 2 (Updating 2-relevant explanations) :
Consider example 1. We assume that no other explanation has been found for the removal $v_1 \neq a$. 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 constraint c_3 is relaxed. That is why both $\{c_1, c_3\}$ and $\{c_2, c_3\}$ are in the $expl(v_1 \neq a, 1)$.

- If constraint c_1 is relaxed: explanation $\{c, c_1, c_2\}$ jumps from $expl(v_1 \neq a, 0)$ to $expl(v_1 \neq a, 1)$; explanation $\{c_1, c_3\}$ is forgotten and thus removed from $expl(v_1 \neq a, 1)$ (we have $k = 2$). The new set of 2-relevant explanations is therefore: $expl(v_1 \neq a, 0) = \{\{c_2, c_4\}\}$, $expl(v_1 \neq a, 1) = \{\{c, c_1, c_2\}, \{c_2, c_3\}\}$.
- If constraint c_3 is reactivated (from the original set of explanations): explanations $\{c_1, c_3\}$ and $\{c_2, c_3\}$ become valid and the new set of 2-relevant explanations is therefore: $expl(v_1 \neq a, 0) = \{\{c, c_1, c_2\}, \{c_2, c_4\}, \{c_1, c_3\}, \{c_2, c_3\}\}$, $expl(v_1 \neq a, 1) = \{\}$.

Computing conflicts The same dilemma that we encountered when computing explanations appears when computing conflicts. Indeed, when a contradiction is identified (a domain of a variable becomes empty), we saw that equation 4 computes a conflict. 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 to valid explanations *i.e.* $expl(v \neq a, 0)$) for all $i \in [0..k-1]$. The resulting number of valid conflicts is:

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

Example 3 (The possible nogoods) :
Let us consider example 2 but with a different k. We assume that values a and b from v_1 are have been removed with the following explanations: $expl(v_1 \neq a, 0) = \{\{c, c_1, c_2\}, \{c_2, c_4\}\}$ and $expl(v_1 \neq b, 0) = \{\{c_1, c_5\}, \{c_2, c_5\}\}$.
Applying equation 5 leads to the following nogoods:

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

Here, the *main* explanation for $v_1 \neq a$ is $\{c, c_1, c_2\}$ and $\{c_1, c_5\}$ is the one for $v_1 \neq b$.

3.2 1-relevance *vs.* classical approaches

All classical approaches (*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[4].

Furthermore, all classical approaches take into account only one conflict. This conflict is computed following to equation 4. Our approach may deal with a *set* of conflicts (see equation 5). Nevertheless that particular explanation management has a computational and spatial cost.

3.3 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:

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

4 An example : the conference problem

To illustrate the use of the k-relevant explanations, we present the resolution of the conference problem [22]. 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 known 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). There domain will be $[1, 2, 3, 4]$ (1 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 $c_5 : Am \neq Pm$.

[4] It will be used to compute conflict. The *main* explanation will still be the only used to compute subsequent explanations.

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 to 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 $\{C_1, C_2, C_3, C_4, C_5, C_6\}$.

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 1. Domains after the introduction of constraints

4.4 Solving using 1-relevant explanations

Table 2 presents the 1-relevant explanations associated to 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 conflicts. In our case, we obtain two conflicts : $\{C_1, C_3, C_4, C_5, C_6\}$ and $\{C_1, C_4, C_5, C_6\}$.

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

4.5 User interaction

As we can see in our example, conflicts and explanations are sets of low-level constraints. Only a specialist can understand and correctly interpret the provided information. In order to design user interaction tools, we therefore need to provide *translation* tools to make accessible to any user the low-level constraints. In the next section, we introduce user-friendly explanations to address that issue.

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**

Table 2. Final set of explanations

5 User-friendly explanations

In this section, we introduce the notion of user-friendly explanations [12]. We provide a set of tools to address the accessibility issue of conflicts and explanations made of low-level constraints.

A solution to that issue will be obtained thanks to the developer of the application. When developing an application, such an expert needs to *translate* the problem from the high level representation (the user's comprehension of the problem) to the low-level representation (the actual constraints in the system). We note this translation a `user` $\rightarrow$ `system` translation.

For user-friendly explanations, we need the other way *translation*: from the low-level constraints (solver adapted) to the user comprehensible constraints (higher level of abstraction). We note this translation a `system` $\rightarrow$ `user` translation. That translation is usually not explicitly coded in the system. Asking a developer to provide such a translator while coding would be quite strange for him. We chose to automatize that translation in a transparent way.

5.1 Hierarchical representation of problems

This idea relies on a single hypothesis: all aspects of a constraint-based application can be represented in a hierarchical way. Indeed, any object appearing in a constraint problem is attached to at most a single *father*-object. Note that a given *father*-object may have several *children*-objects. For example, figure 1 gives a graphical representation of the problem defined in section 4.1.

5.2 Building a `system` $\rightarrow$ `user` translator

While developing a constraint application, the user only needs to explicitly state the underlying hierarchy of his problem. Only the leaves of this structure, namely the low-level constraints, can be used by the constraint solver.

The leaves may be too low-level for a typical user of the final application. However, he can understand higher levels in the hierarchy. What allows the hierarchy hypothesis is to build with no effort for the developer a hierarchical representation of the problem. Once built, this representation can be used to interact with any user through user-friendly explanations. Such explanations are provided using procedures converting low-level constraints into user understandable pieces of the hierarchy. Those procedures are completely problem independent and may be provided within the constraint solver.

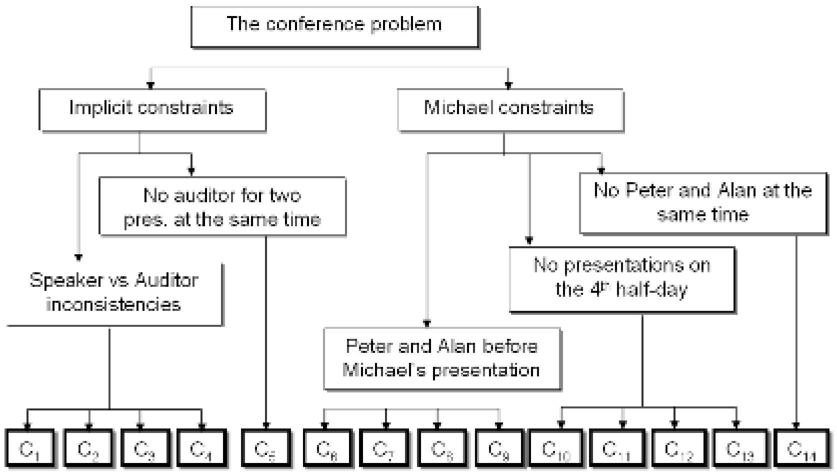
The conference problem
Implicit constraints
Michael constraints
No auditor for two pres. at the same time
Speaker vs Auditor inconsistencies
No Peter and Alan at the same time
No presentations on the 4th half-day
Peter and Alan before Michael's presentation
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
C13
C14

Fig. 1. An hierarchical view of the conference problem

The user perception of a given problem can be seen as a *cut* in the hierarchical view of the considered problem. For example, let suppose that the user is Michael who does not want to deal with implicit constraints. Although he does understand his own wishes. Therefore, his view of the problem would be: $\boxed{\text{The conf. problem}}$, $\boxed{\text{P\&A before}}$, $\boxed{\text{Not } 4^{th} \text{ 1/2 day}}$ and $\boxed{\text{P\&A not same time}}$.

Our example has no solution, one explanation for that situation provided by a classical approach is: $\{c_1, c_2, c_3, c_4, c_5, c_6\}$.

Michael looks at the explanation from his point of view. We therefore need to project the concrete explanation onto his representation. The projection consists in projecting the constraints $\{c_1, c_2, c_3, c_4, c_5, c_6\}$ from bottom to the top of the hierarchy representing the problem (see figure 1) until a user understandable box is reached.

For example, the projection of the constraints $\{c_1, c_2, c_3, c_4\}$ gives in first step the $\boxed{\text{Speaker vs. Auditor}}$ box. Unfortunately, this box is not understandable by Michael. In this case, the projection continues to the father box: $\boxed{\text{Implicit constraints}}$. Once again, this box is not understandable by Michael and the projection gets to $\boxed{\text{The conf. problem}}$. Finally, we reached $\boxed{\text{The conf. problem}}$ box which Michael can deal with. The projection of c_5 gives the same box through $\boxed{\text{Auditor vs. 2 pers.}}$ and $\boxed{\text{Implicit constraints}}$. For c_6, the projection is easier because the first reached box is user understandable.

The final projection gives: $\boxed{\text{The conf. problem}}$ and $\boxed{\text{P\&A before}}$. He can ask to discard one box. For example, the box *Michael wants to speak after Peter and Alan.*

6 Exploiting *k*-relevant explanations

k-relevance provides more interesting explanations and allows to obtain a better diagnosis. In this section, we present five concrete situations which the user is frequently confronted in the case of a failure. We show how *k*-relevant explanations allow to better understand and to analyze this failure and to quickly simulate various scenarios.

6.1 Giving 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 $e_2 - e_1$ are not responsible of 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). But, the multiplicity of explanations leads to more precise explanations and conflicts.

6.2 Does a constraint belong to a conflict?

An interesting issue when debugging is to know wether a given constraint belongs to a conflict or not. k-relevant explanations help answering that question.

Let 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* by referring to the table 3. Based on the classical approach, the only conflict will 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 conflicts and indicates that the constraint C_5 is strongly responsible of the incoherence.

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

Table 3. New state of the variable A_m

6.3 Simulating constraint relaxation

Determining if a given constraint belongs to a conflict or not may lead to spend 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 suppose that the user suspects that constraint C_3 belongs to a conflict. The constraint-checking tool (see section 6.2) confirms it. The relaxation of this constraint will put back all the values a such as $C_3 \in \mathit{expl}(A_m \neq a)$. 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 towards an another failure due to the removal $A_m \neq 1$ that will be justified by another explanation: $\{C_5\}$. Thus, our tool is able to indicate to the user if a relaxation of a suspected constraint will lead to another *immediate* failure.

6.4 Simulating constraint addition

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

It is helpful to take advantage of the information accumulated during the resolution of the previous problem to avoid adding constraints leading towards an *immediate* failure. For this reason, we have proposed 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 towards a failure or not. So we can avoid reintroducing such constraints.

For example, let suppose now that the user has removed constraints $\{c_3, c_5\}$ to put back value 1 in the domain of A_m. The new domains (along with the associated explanations) are reported in table 4. Let suppose that our user wants to put back the previously relaxed constraint c_3. A naive approach consists in propagating the constraint from the current situation (it leads to a contradiction). However, 2-relevant explanations can be used to simulate this constraint addition by updating the relevance status of associated explanations. Some of them may become valid ($\{c_3\}$ in our example) and therefore remove some values (1 from A_m here). Here, with no propagation at all, using k-relevant explanation could have helped the user by telling him that adding constraint c_3 would have lead to a contradiction.

Variable	Value	Explanation	1-relevance	2-relevance	present ?
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

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

6.5 Providing error diagnosis

Imagine now that after some relaxations, the user wants to know why the 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 notice that 1-relevance provides a richer diagnosis than the classical approach.

7 Implementation

COINS has been implemented in `choco`[5] [2] using the `PaLM` system [8]. `choco` allows : to propagate the constraints, to manage domains as well as other filtering algorithms, local search, etc. Experiments show that when k increases, the performance of k-relevance decreases. In the other hand, for $k = 1$ and $k = 2$, we obtain the same temporal performances as **mac-dbt**[15]. For $k = 1$ and $k = 2$, the time lost to manage the k-relevant explanations is compensated by avoiding future failures. For $k \geq 3$, k-relevance loses its advantages. More precisely, for such a k, we are losing time managing explanations that will seldom let us avoid future failures. Especially, we shall update explanations which will never become valid (*i.e.* 1-relevant).

[5] `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.

8 Conclusion

In this paper, we have introduced the foundations of several interactive tools which are of great help for a user to solve an over-constrained problem. We have shown the effectiveness of these k-relevance-based tools. k-relevance keeps several explanations by removal and forgets them once they become irrelevant.

We have shown the contribution of our k-relevance-based tools compared to classical approach. k-relevance provides more precise explanations; gives some general information that cannot be 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 from all the explanations. We investigate adding user-based comparators [23] to our tools in order to provide automatic comparison of solutions. Also, we try to decrease the space complexity by managing differently 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: 8th 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. Bayardo Jr., R.J., Miranker, D.P.: A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In: AAAI'96. (1996)
10. Schiex, T., Verfaillie, G.: Nogood Recording fot Static and Dynamic Constraint Satisfaction Problems. International Journal of Artificial Intelligence Tools **3** (1994) 187–207
11. Jussien, N.: e-constraints: explanation-based constraint programming. In: CP01 Workshop on User-Interaction in Constraint Satisfaction, Paphos, Cyprus (2001)
12. Jussien, N., Ouis, S.: User-friendly explanations for constraint programming. In: ICLP'01 11th Workshop on Logic Programming Environments, Paphos, Cyprus (2001)
13. 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
14. Ginsberg, M.: Dynamic backtracking. Journal of Artificial Intelligence Research **1** (1993) 25–46
15. 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
16. Bliek, C.: Generalizing partial order and dynamic backtracking. In: Proceedings of AAAI. (1998)
17. 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).
18. Prosser, P.: MAC-CBJ: maintaining arc-consistency with conflict-directed backjumping. Research Report 95/177, Department of Computer Science – University of Strathclyde (1995)
19. Glover, F., Laguna, M.: Modern heuristic Techniques for Combinatorial Problems, chapter Tabu Search, C. Reeves. Blackwell Scientific Publishing (1993)
20. Schiex, T., Verfaillie, G.: Nogood recording for static and dynamic CSP. In: 5th IEEE International Conference on Tools with Artificial Intelligence, Boston, MA. (1993) 48–55

21. Dechter, R.: Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. Artificial Intelligence **41** (1990) 273–312
22. 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
23. 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