k-relevant Explanations for Constraint Programming

Samir Ouis and **Narendra Jussien**
École des Mines de Nantes – BP 20722
F-44307 NANTES Cedex 3 – FRANCE
Samir.Ouis@emn.fr
Narendra.Jussien@emn.fr

Patrice Boizumault
GREYC, CNRS UMR 6072
Université de Caen, Campus 2,
F-14032 CAEN Cedex – FRANCE
Patrice.Boizumault@info.unicaen.fr

Abstract

This paper presents diagnosis tools and interaction-based tools which could help the Constraint Programming user to interactively develop its applications. The implementation of these tools rely on explanations, and more precisely on k-relevant explanations (Bayardo Jr. & Miranker 1996). An example is given to illustrate k-relevant explanations and to provide concrete situations illustrating the functionalities of our interactive and diagnosis tools.

Introduction

Constraint programming (CP) 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 have been developed and widely spread: CHIP, CHOCO, GNUPROLOG, ILOG SOLVER, etc. But these systems are helpless when the constraints network to solve has no solution. Indeed, the user is left alone to seek why the solver answered her `no solution`: 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 (Junker 2001) computes conflict-sets for configuration problems, (Bessière 1991) and (Debruyne 1996) introduce tools to dynamically remove constraints, PALM (Jussien 2001) uses conflict-sets to address those issues and defines new search algorithms, (Squali & Freuder 1996) introduces constraint-specific tools for providing user-friendly solutions to constraint problems, (Freuder, Likitvivatanavong, & Wallace 2000) 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 (Bayardo Jr. & Miranker 1996). The idea is to record bounded sets of explanations (Jussien 2001) (rather than a single one) based on their relevance. An explanation is said to be k-relevant if it contains less than k constraints that are not valid *w.r.t.* the current store of constraints (1-relevant explanations are valid ones).

(a measurement of the distance between the current situation and a stored explanation). This relevance-based long term memory for explanations is used to design interaction-based tools, diagnosis tools as well as improved search techniques.

This paper is organized as follows: first, we recall the definition of conflict-sets and explanations and discuss their storing. Then, we introduce k-relevant explanations and give an example. Next, we show how k-relevant explanations are well suited for building both diagnosis tools and interaction-based tools before concluding and presenting some further works.

Conflict-sets and explanations for CP

A *Constraint Satisfaction Problem* (CSP) is defined by a set of variables $V = \{v_1, v_2, \ldots, v_n\}$ taking their values in their respective domains $D = \{d_1, \ldots, d_n\}$ 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, D, 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*.

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** (Schiex & Verfaillie 1994)) 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: a subset of the original set of constraints ($C' \subset C$ in the following equation) and a subset of decision constraints introduced so far in the search (here $dc_1, \ldots, dc_k$) *i.e.* $\mathbf{sol}\,(V, (C' \wedge dc_1 \wedge ... \wedge dc_k)) = \emptyset$.

An operational viewpoint of conflict-sets can be made explicit by rewriting the previous equation the following way: $C' \wedge \left(\bigwedge_{i \in [1..k] \setminus j} dc_i\right) \rightarrow \neg dc_j$.

If $dc_j : v_j = a$ in the previous formula, the previous result

can be read as ($s(v)$ is the value of variable v in the solution s): $\forall s \in \mathbf{sol}\left(V, C' \wedge \left(\bigwedge_{i\in[1..k]\setminus j} dc_i\right)\right), s(v_j) \neq a$.

The left hand side of the previous implication 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 previous ones. Notice that conflict-sets can be computed from explanations for the empty domain of a variable v: $\bigwedge_{a\in d(v)} \texttt{expl}(v \neq a)$.

There generally exists several explanations for the removal of a given value. Several different approaches were introduced to handle that multiplicity: from a complete storing (like in *Dependency Directed Backtracking* (Stallman & Sussman 1977)) leading to an exponential space complexity to storing only a single explanation (like in *Dynamic Backtracking* (Ginsberg 1993) and its improvements or in *Conflict-directed BackJumping* (Prosser 1995)). The idea, here, is to erase (*i.e.* forget) explanations as soon as they are no longer valid with the current set of decision constraints. Space complexity remains polynomial while keeping the algorithms complete. However, such a drastic behavior is not compatible with developing efficient debugging tools: a large part of the search history will be completely lost.

Between recording all and only one explanation, an interesting idea is to pick up a criterion that will be used to erase/forget past explanations. It can either be: (a) a *time-bounded criterion*: explanations are forgotten after a given time (this criterion is similar to *tabu* list management in *tabu* search (Glover & Laguna 1993)); (b) a *size-bounded criterion* (Schiex & Verfaillie 1994): only explanations having a size lower or equal to a given value n are kept (this criterion limits the spatial complexity, but may forget really interesting conflict-sets); or (c) a *relevance-bounded criterion*: explanations are kept if they are still close to the current set of decision constraints. This last concept (called k-relevance) has been introduced in (Bayardo Jr. & Miranker 1996) and focuses explanations/conflict-sets management to what is important: relevance *w.r.t.* the current situation. Time and size-bounded recording do have a controllable space complexity. It will be the same for k-relevance learning. As we shall see, our tools are meant for debugging and interactive development of CP programs: the space occupation overhead (compared to recording-free techniques) is well worth it.

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 (Bayardo Jr. & Miranker 1996), we can now formally define a k-relevant explanation as:

Definition 1 *k-relevant explanation*
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 only a single explanation but the set of currently k-relevant explanations recorded for the removal of value a from the domain $d(v)$ of variable v.

Computing k-relevant explanations k-relevant explanations, as regular explanations (Jussien 2001), 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 (it is called the **main** one). Only that explanation will be used to compute forthcoming explanations[1]. Moreover, this explanation is exactly the one that would have been computed by a classical approach. 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 $\{c, c_1, c_2\}$.

We need to maintain the relevance information attached to stored explanations upon constraint additions and retractions. In both ways, the relevance of some explanations may evolve. The idea is to keep track of these variations and to forget explanations as soon as they become irrelevant (their relevance is greater than k). 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.

[1] 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* k-relevant explanations.

The dilemma encountered when computing explanations appears again for **computing conflict-sets**. Indeed, when a contradiction is identified (the domain of a variable becomes empty), we saw above how to compute a conflict-set. However, there may exist several explanations for each 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 $a \in d(v)$. The resulting number of valid conflict-sets is: $\prod_{a \in d(v)} |expl(v \neq a, 0)|$

Complexity issues Let us consider a CSP with 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: $\left(\begin{array}{c} e+n+k-1 \\ (e+n+k-1)/2 \end{array} \right)$ subsets of size $(e+n+k-1)/2$.

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

$$O\left(n \times d \times \left(\begin{array}{c} e+n+k-1 \\ (e+n+k-1)/2 \end{array} \right) \times (e+n+k-1)/2\right)$$

Discussion

- **Classical approaches vs 1-relevance**
  All classical approaches (*eg. Dynamic Backtracking* or MAC-DBT (Jussien, Debruyne, & Boizumault 2000)) 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 value 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[2]. Furthermore, all classical approaches take into account only one conflict-set. 1-relevance will generally have to deal with more than one conflict-set. Nevertheless that particular explanation management has a computational and spatial cost.
- **How to compute the best conflict-set?**

  The most interesting conflict sets are those which are minimal regarding inclusion. Minimal ones can be obtained by computing a covering of all the valid explanations. Unfortunately, computing such a set is exponentially costly. The simplest conflict-set can be computed by taking the union of all the valid explanations; but thus a lot of precision will be lost. A good compromise between both precision and ease of computation is to select only one explanation by value removal (namely the main one) and then to union them in order to build the conflict-set. Instead of always taking the main explanation for a value removal, the selected explanation could be chosen by a comparator which will be able to take the user's preferences into account.

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

An example : the conference problem

To illustrate the use of the k-relevant explanations, we present the resolution of the conference problem (Jussien & Boizumault 1996). From now on, we fill focus our study to 1-relevance.

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.

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 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: $Ma \neq Am$, $Mp \neq Pm$, $Ma \neq Pm$ and $Mp \neq Am$.
- No one can attend two presentations at the same time. This is modelled as $c_1 : Am \neq Pm$.

Michael constraints can be modelled:

- Michael wants to speak after Peter and Alan: $c_2 : Ma > Am$, $c_3 : Ma > Pm$, $c_4 : Mp > Am$ and $c_5 : Mp > Pm$.
- Michael does not want to come on the fourth half-day: $c_6 : Ma \neq 4$, $c_7 : Mp \neq 4$, $c_8 : Am \neq 4$ and $c_9 : Pm \neq 4$.
- Michael does not want to present to Peter and Alan at the same time: $c_{10} : Ma \neq Mp$.

Table 1 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.

Table 1: Final set of explanations

Var	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**

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 to in the case of failure. We show how k-relevant explanations enable to build more efficient interactive and diagnosis tools.

Diagnosis tools

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

Providing more precise explanations If there are two explanations for the same removal e_1 and e_2 such as $e_1 \subsetneq e_2$, then constraints which belong to $e_2 \setminus e_1$ are not responsible for that 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 2). 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 2. Based on the classical approach, the alone conflict-set would be $\{c_2, c_3, c_4, c_6\}$. Indeed, the answer will be negative ($c_5 \notin \{c_2, c_3, c_4, c_6\}$). 1-relevant explanations provide 8 conflict-sets and indicate that constraint c_5 is strongly responsible for the incoherence (it removes 3 values out of 4 in A_m).

Table 2: New state of the variable A_m

Var	Value	Explanation	1-relevance	present
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? Classical explanations provide the explanation $\{c_6\}$, while 1-relevant explanations give a more precise set of explanations: $\{\{c_4\}, \{c_5\}, \{c_6\}\}$. 1-relevance provides a better 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 constraints are very far from the end-user's vision of the solved problem. So, we have introduced in (Jussien & Ouis 2001) a way of providing user-readable explanations by using a tree-based representation of the user's understanding of the solved problem.

Interaction-based tools

k-relevance allows the simulation of constraint addition/retraction with a negligible computational cost.

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 and that the constraint-checking tool 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 2, 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 to know that the relaxation of constraint c_3 will lead to 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 relaxing a given constraint will lead to another *immediate* failure with a neglectful computational cost.

Simulating constraint addition To solve a dynamic problem, re-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 the addition of a previously relaxed constraint will lead to a future immediate failure.

Table 3: New state of the variable A_m after relaxation of c_3 and c_5

Var	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

It is helpful to take advantage of the information stored during the resolution of the previous problem in order to avoid adding constraints leading to *immediate* failures. For this reason, we propose a tool simulating the re-introduction of an already relaxed constraint without any propagation. This tool will tell the user if the addition of a relaxed constraint will lead 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 (see table 3). Later, the user wants to put back the relaxed constraint c_3. The classical approach would have put back the constraint c_3 and naively launched the propagation phase leading to a contradiction. However, the k-relevance approach can simulate this constraint comeback by updating the 2-relevant 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 tells 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.

Improving explanation-based search algorithms

We have also experimented the use of k-relevant explanations for explanation-based search algorithms such as MAC-DBT (Jussien, Debruyne, & Boizumault 2000). 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. Precisely, for $k = 1$ and $k = 2$, the time required to manage k-relevant explanations is compensated by the time gained by avoiding failures. For $k \geq 3$, too much time is spent for managing explanations which will seldom let us avoid future failures. Especially, we will have to update explanations which could never become valid (*i.e.* 1-relevant).

Therefore, a good compromise would be to use $k = 1$ or $k = 2$, even if we will have to pay a small penalty for 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.

Conclusion

In this paper, we have proposed the foundations of several interactive tools which are of great help for a user to develop CP applications. We have shown the effectiveness of k-relevance explanations for building interactive and diagnosis tools. This effectiveness comes from the fact that k-relevance provides more numerous and precise explanations.

Notice that (Amilhastre, Fargier, & Marquis 2002) are also interested in the design of interactive and diagnosis tools in decision support systems for configuration problems. There are two main differences with our proposal: i) the CSP representing the initial problem is considered to be persistent and compiled into an automaton, ii) as a consequence, a user can only interact by retracting/adding decision constraints. Our proposal does not impose such a restriction.

Our current work includes designing algorithms which can compute efficiently *best* conflict-sets. For this, we plan to introduce user-based comparators (Borning *et al.* 1989) in order to compare solutions. Also, we are trying to decrease the space required to manage k-relevant explanations.

References

Amilhastre, J.; Fargier, H.; and Marquis, P. 2002. Consistency restoration and explanations in dynamic csps - application to configuration. *Artificial Intelligence* 135(2002):199–234.

Bayardo Jr., R. J., and Miranker, D. P. 1996. A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In *AAAI'96*.

Bessière, C. 1991. Arc consistency in dynamic constraint satisfaction problems. In *Proceedings AAAI'91*.

Borning, A.; Maher, M.; Martindale, A.; and Wilson, M. 1989. Constraint hierarchies and logic programming. In Levi, G., and Martelli, M., eds., *ICLP'89: Proceedings 6th International Conference on Logic Programming*, 149–164. Lisbon, Portugal: MIT Press.

Debruyne, R. 1996. Arc-consistency in dynamic CSPs is no more prohibitive. In 8^{th} *Conference on Tools with Artificial Intelligence (TAI'96)*, 299–306.

Freuder, E. C.; Likitvivatanavong, C.; and Wallace, R. J. 2000. A case study in explanation and implication. In *In CP2000 Workshop on Analysis and Visualization of Constraint Programs and Solvers*.

Ginsberg, M. 1993. Dynamic backtracking. *Journal of Artificial Intelligence Research* 1:25–46.

Glover, F., and Laguna, M. 1993. *Modern heuristic Techniques for Combinatorial Problems, chapter Tabu Search, C. Reeves.* Blackwell Scientific Publishing.

Junker, U. 2001. QUICKXPLAIN: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints.*

Jussien, N., and Boizumault, P. 1996. Implementing constraint relaxation over finite domains using ATMS. In Jampel, M.; Freuder, E.; and Maher, M., eds., *Over-Constrained Systems*, number 1106 in Lecture Notes in Computer Science, 265–280. Springer-Verlag.

Jussien, N., and Ouis, S. 2001. User-friendly explanations for constraint programming. In *ICLP'01 11th Workshop on Logic Programming Environments (WLPE'01).*

Jussien, N.; Debruyne, R.; and Boizumault, P. 2000. Maintaining arc-consistency within dynamic backtracking. In *Principles and Practice of Constraint Programming (CP 2000)*, number 1894 in Lecture Notes in Computer Science, 249–261. Singapore: Springer-Verlag.

Jussien, N. 2001. e-constraints: explanation-based constraint programming. In *CP01 Workshop on User-Interaction in Constraint Satisfaction*.

Prosser, P. 1995. MAC-CBJ: maintaining arc-consistency with conflict-directed backjumping. Research Report 95/177, Department of Computer Science – University of Strathclyde.

Schiex, T., and Verfaillie, G. 1994. Nogood Recording fot Static and Dynamic Constraint Satisfaction Problems. *International Journal of Artificial Intelligence Tools* 3(2):187–207.

Squali, M. H., and Freuder, E. C. 1996. Inference-based constraint satisfaction supports explanation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI '96)*, volume 1, 318–324. Portland, Oregon.

Stallman, R. M., and Sussman, G. J. 1977. Forward reasoning and dependency directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence* 9:135–196.