Available online at www.sciencedirect.com
ScienceDirect
Applied Soft Computing 8 (2008) 1400–1406
Applied Soft Computing
www.elsevier.com/locate/asoc

An explanation-based tools for debugging constraint satisfaction problems

Samir Ouis [a,*], Mohamed Tounsi [b]

[a] Imam University, Computer Science Department, Riyadh, P.O. Box 84880, Saudi Arabia
[b] Prince Sultan University, Computer Science Department, Riyadh, P.O. Box 66833, Saudi Arabia

Received 6 September 2005; received in revised form 7 October 2007; accepted 16 October 2007
Available online 17 November 2007

Abstract

This paper describes an explanation-based tools for constraint programming system. These tools provide to the user the conflicts when it raise during solving process. Our tools simulate constraint additions and/or constraint relaxations without any propagation; it also determine if a given constraint belongs to a conflict and it provide diagnosis tool (*e.g.* why variable *v* cannot take value *val*?). With more user-friendly representation of conflicts and explanations, our proposed tools give better problem understanding to the user. We prooved that the proposed tools are efficient, and while there is no debugging system that allow the user to interact with the solver, our explanation-based tools could be used for many other applications.

Keywords: Constraint programming; Explanations; Conflicts; Debugging; Solver

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 [6–8] introduces tools to dynamically remove constraints, etc.

User interaction requires user-friendly and interactive tools. In this paper, we advocate for the use of *k-relevant* explanations [9] to provide explanation-based solving system.

The proposed system helps the user to understand inconsistency, to simulate constraint additions and/or retractions (without any propagation), and to determine if a given constraint belongs to a conflict. The most important feature is that it provides diagnosis tools (*e.g.* why variable *v* cannot take value *val*?); in fact our system is based upon the use of conflict sets (*a.k.a.* nogood [10]), explanations [11].

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*-relevant-bounded explanations (Section 3) and give an illustrative example (Section 4). After, we illustrate the use of *k*-relevant explanations in our system (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).

* Corresponding author.
E-mail addresses: sam_ouis@hotmail.com (S. Ouis), mtounsi@cis.psu.edu.sa (M. Tounsi).

1568-4946/$ – see front matter
doi:10.1016/j.asoc.2007.10.019

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[1]: a subset of the original set of constraints ($C' \subset C$ in Eq. (1)) and a subset of decision constraints introduced so far in the search (here $dc_1, \ldots, dc_k$).

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

An operational viewpoint of conflict sets can be made explicit by rewriting Eq. (1) the following way:

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

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

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

The left-hand-side of the implication in Eq. (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 $dc_1 \bigvee \ldots \bigvee dc_j$ 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):

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

[1] 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 $C' = \varnothing$, 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.

2.2. Storing explanations: k-relevant-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* [12] 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* [13] and its improvements (*MAC-DBT* [14], *generalized dynamic backtracking* [15], *partial-order dynamic backtracking* [16]) and *conflict-directed backjumping* [17]. 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 [18].
- Size-bounded criterion: [19] has used a criteria defined in [20]. 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.
- Relevant-bounded criterion: explanations are kept if they are not *too far* from the current set of decision constraints. This concept (called k-relevant) has been introduced in [9] and focus explanations/conflict management to what is important: relevanct wrt the current situation.

Time and size-bounded recording do have a controllable space complexity. This is also the case for k-relevant 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-Relevant-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.* $|e \cap R| < k$.