Artificial Intelligence 238 (2016) 135–153

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Learning general constraints in CSP

Michael Veksler, Ofer Strichman *

Information Systems Engineering, IE, Technion, Haifa, Israel

ARTICLE INFO

Article history:
Received 5 July 2015
Received in revised form 3 June 2016
Accepted 9 June 2016
Available online 14 June 2016

Keywords:
Constraints solving
Inference rules

ABSTRACT

We present a new learning scheme for solvers of the Constraint Satisfaction Problem (CSP), which is based on learning (general) constraints rather than the generalized no-goods or signed-clauses that were used in the past. The new scheme is integrated in a conflict-analysis algorithm reminiscent of a modern systematic propositional satisfiability (SAT) solver: it traverses the conflict graph backwards and gradually builds an asserting conflict constraint. This construction is based on new inference rules that are tailored for various pairs of constraints types, e.g., $x \leq y_1 + k_1$ and $x \geq y_2 + k_2$, or $y_1 \leq x$ and $[x, y_2] \not\subseteq [a, b]$. The learned constraint is stronger than what can be learned via signed resolution. Our experiments show that our solver HCSP backtracks orders of magnitude less than other state-of-the-art solvers, and is overall on par with the winner of this year's MiniZinc challenge.

1. Introduction

Numerous industrial and academic decision and/or optimization problems can be modeled naturally as a Constraint Satisfaction Problems (CSP) [1]. Indeed, problems in scheduling, planning, verification, test generation and more are solved regularly with such tools. The common definition of this problem refers to a set of variables over finite and discrete domains, and a set of constraints over these variables. The decision variant of the CSP (we only refer to this variant in the article and ignore the optimization variant) is to find whether there exists an assignment to the variables from their respective domains that satisfies all the constraints, and emit such an assignment as output if indeed such an assignment exists. For example, we can define five variables $x_1, \ldots, x_5$, each with the domains $\{1..5\}$ and ask whether the constraint $\texttt{AllDifferent}\ (x_1, \ldots, x_5) \wedge x_3 < x_1$ is satisfiable, where `AllDifferent` is a constraint requiring its input variables to be assigned different values. There is a common set of constraints that is supported by many CSP solvers called FlatZinc [2], which includes over a hundred constraint types, including Boolean connectives. The decision variant of the CSP is NP-complete. While theoretically it has the same expressive power as propositional satisfiability (SAT) [3], its rich modeling language leads to easier, more succinct modeling, and furthermore, certain types of constraints can be solved efficiently (polynomially) whereas solving their translation to propositional logic is exponential. In practice the benefit of these two differences is not always apparent, because frequently the modeling is done once and then instances are generated automatically and do not need to be readable; furthermore, in practice almost all CSPs combine many types of constraints that at least some of them cannot be solved efficiently, hence the NP-hardness of the problem is evident in practice. The tremendous progress in efficient SAT solving as of the introduction of the SAT solver Chaff [4] in 2001 has led to the development of several competitive CSP solvers (at the time they were developed) that translate the input CSP to SAT, e.g., see [5].

* Corresponding author.
E-mail addresses: mveksler@tx.technion.ac.il (M. Veksler), ofers@ie.technion.ac.il (O. Strichman).

http://dx.doi.org/10.1016/j.artint.2016.06.002
0004-3702/

The ability of CSP solvers to *learn* new constraints during the solving process possibly shortens run-time by an exponential factor (see, e.g., [6]). Learning in a limited form was present in early CSP solvers, where it was called *nogood learning* [7]. Nogoods are defined as partial assignments that cannot be extended to a full solution. Later *generalized nogoods* [6] (g-nogoods for short) were proposed, which allow *non-assignments* as well, e.g., a g-nogood $(x \nleftarrow 1, y \leftarrow 1)$ means that an assignment in which x is assigned anything but 1 and y is assigned 1 cannot be extended to a solution. This formalism is convenient for representing knowledge obtained during the search for a solution. The g-nogood above, for example, can result from removing 1 from the domain of x, which leads the solver to remove 1 from the domain of y. G-nogoods may be exponentially stronger than nogoods, as shown in [6]. Another extension is c-nogoods [8], which are like g-nogoods, but a literal can reify a general constraint. c-nogoods are implemented in MINION [9] for a few types of constraints.

A more general and succinct representation of learned knowledge is in the form of *signed clauses*. Such clauses are disjunctions of *signed literals*, where a signed literal has the form $v \in D$ or $v \notin D$ (called positive and negative signed literals, respectively), where v is a variable and D is a domain. Beckert et al. [10] studied the satisfiability problem of signed CNF, i.e., satisfiability of a conjunction of signed clauses. Their inference system is based on simplification rules and a rule for binary resolution of signed clauses:

$$\frac{((v \in A) \vee X) \quad ((v \in B) \vee Y)}{(v \in (A \cap B) \vee X \vee Y)} \quad [\text{Signed Resolution}(v)] \tag{1}$$

where X and Y consist of a disjunction of zero or more literals, A and B are sets of values, and v is called the *pivot* variable. Note that in case v is Boolean and A, B are complementary Boolean domains (e.g., $A = \{0\}, B = \{1\}$) then this rule simplifies to the standard resolution rule for propositional clauses that is used in SAT, namely the consequent becomes $(X \vee Y)$.

As we showed in [11], we used this rule in our CSP solver HCSP (short for HaifaCSP),[1] as part of a general learning scheme based on signed clauses. Using a special inference rule for each type of non-clausal constraint, HCSP inferred a signed clause e that *explains* a propagation by that constraint. This means that e is implied by the constraint, but at the same time is strong enough to make the same propagation as the constraint, at the same state. Using such explanations in combination with rule (1) for resolving signed clauses, HCSP can generate a signed *conflict clause* via *conflict analysis*. By construction this clause is *asserting* (i.e., it necessarily leads to additional propagation after backtracking). In contrast to the CSP solver EFC [6], which generates a g-nogood *eagerly* for each removed *value*, HCSP generates a signed explanation clause *lazily*, only as part of conflict analysis. Lazy learning of g-nogoods was also implemented on top of MINION [9]. In [12] propositional explanations are generated for highly active constraints. There has also been work on extending explanations with new Boolean variables, which encode equalities and inequalities [13,14], lazy model expansion [15] where the formula is lazily grounded (i.e., not related to conflicts), learning non-ground rules in the context of answer-set programming [16], and constraint-specific inference [17], such as partial sums in the case of linear constraints. In all these works there is no direct inference between general constraints.

In this article we study a different learning scheme, which is based on inference rules with non-clausal consequents. Non-clausal learning has been studied before in the context of several first-order quantifier-free theories: Pseudo-Boolean constraints (see, e.g., Sect. 22.6.4 in [18] and [19]), difference constraints [20], and integer linear constraints, e.g., [21,22]. The congruence-closure algorithm for equality logic with uninterpreted functions, which is implemented in most SMT solvers, can also be seen as inferring non-clausal constraints, since it infers new equalities. In all of these cases such learning was shown to improve the search, which motivated us to develop such a scheme for CSP, that is strongly tied to the conflict-analysis procedure. What we suggest here is very general, as it can be used with most of the constraints that are supported by modern CSP solvers, and allows non-clausal inference between different types of constraints.

Our main goal in introducing this scheme is to learn a conflict constraint that is logically stronger and not harder to compute than its clausal counterpart. The emphasis is on the first of these goals as it may improve the search itself. To that end, we propose a generic inference rule called *Combine* that for many popular (pairs of) constraints indeed fulfills these two goals. For example, suppose that a CSP has Boolean variables x, y_1, y_2, y_3, and two constraints ($\oplus$ denotes XOR)

$$c_1 \doteq x \oplus y_1 \oplus y_3 = 0 \qquad c_2 \doteq x \oplus y_2 \oplus y_3 = 0\,.$$

At a state defined by

$$x \in \{0,1\} \quad y_1 \in \{1\} \quad y_2 \in \{0\} \quad y_3 \in \{0\}\,,$$

c_1 propagates $x \in \{1\}$, and then c_2 detects a contradiction. Without going into the details of how clausal explanation works (this will be the subject of Sect. 2.3), in this case it produces the explanation clause

$$(y_1 \in \{0\} \vee y_2 \in \{1\} \vee y_3 \in \{1\})$$

[1] In [11] it was still called PCS, for Proof-producing Constraint Solver.