

Greater Efficiency for Conditional Constraint Satisfaction

Mihaela Sabin¹, Eugene C. Freuder², and Richard J. Wallace²

¹ Department of Mathematics and Computer Science
Rivier College, 420 Main Street, Nashua, NH 03060, USA
`msabin@rivier.edu`

² Cork Constraint Computation Center, Department of Computer Science
University College Cork, Cork, Ireland
`{e.freuder,r.wallace}@4c.ucc.ie`

Abstract. A conditional constraint satisfaction problem (CCSP) extends a standard constraint satisfaction problem (CSP) with a condition-based component that controls what variables participate in problem solutions. CCSPs adequately represent configuration and design problems in which selected subsets of variables, rather than the entire variable set, are relevant to final solutions. The only algorithm that is available for CCSP and operates directly on the original, unreformulated CCSP statement has been basic backtrack search. Reformulating CCSPs into standard CSPs has been proposed in order to bring the full arsenal of CSP algorithms to bear. One reformulation approach adds null values to variable domains and transforms CCSP constraints into CSP constraints. However, a complete null-based reformulation of CCSPs has not been available. In this paper we provide more advanced algorithms for CCSP and a full null-based reformulation into standard CSP. Thorough testing reveals that the advanced algorithms perform up to two orders of magnitude better than plain backtracking, but that realizing practical advantages from reformulation is problematic. The advanced algorithms extend forward checking and maintaining arc consistency to CCSPs. The null-based reformulation improves on the preliminary findings in [1] by removing the limitation on multiple activation, and by localizing changes. It identifies and addresses a difficulty presented by activity cycles.

1 Introduction

There are many important and complex tasks to which constraint satisfaction has been successfully applied. As a result, specialized constraint satisfaction problem (CSP) classes have emerged to cope more directly with specific characteristics of various application domains. Qualifiers such as partial, dynamic, hierarchical, composite, interval, mixed, and others characterize CSP specializations that have been studied in the last decade. The conditional constraint satisfaction is another specialization developed to cope with the special features of diagnosis and configuration problems.

Conditional CSP extends standard CSP with a condition-based component that models dynamic changes of problem solutions with predefined conditions. The formalism has been introduced in [2] under the name of dynamic CSP. It integrates classical constraint satisfaction with a special type of constraint, *activity constraints*, responsible for selecting those variables that should participate in solutions. The formalism has been originally motivated by synthesis tasks such as product configuration, in which not all cataloged components are present in every single configured product. This class of dynamic CSPs is renamed *conditional constraint satisfaction problems (CCSPs)* [3] to (1) capture the nature of the control component that conditionally changes the initial model of the problem, and to (2) distinguish this class of problems from another class of dynamic CSPs that reuses problem solutions when problem changes over time [4,5,6].

Since its first formalization in 1990, conditional constraint satisfaction paradigm has been used for modeling not only configuration problems, but also diagnosis [7], design [1], and network management [8] problems. Despite increasing interest in the area of representing application problems as CCSPs, little progress has been made in the area of improving direct solving methods that operate on CCSP representations. In contrast with other CSP specializations, no standard CSP solving method, except for backtrack search [1], has been adapted to the conditional domain. The lack of specialized, direct solving methods is compounded by the fact that a benchmark test base for this type of problems is extremely limited [9,10], although very much needed in experimental evaluations.

In this paper we present two advanced methods for solving CCSPs that use local consistency methods of forward checking and maintaining arc consistency. Solving methods find values for the set of active variables. These are obtained from the initial set of variables that are assigned values in every solution, and variables that are newly incorporated into the problem via activity constraints. The technical challenges encountered and overcome in extending forward checking and maintaining arc-consistency to CCSP are to: (1) keep track of variables' activity status as determined by consistency checking of activity constraints, (2) enforce chosen level of consistency when checking both compatibility and activity constraints, (3) in case of maintaining arc consistency, extend arc consistency with activation consistency along activity constraints.

The opportunity of importing efficient standard algorithms, whose behavior has been extensively tested, raises new challenges. Are there available similarly comprehensive experimental studies for evaluating CCSP solving? The reality of many application domains, such as configuration or diagnosis, is that either real-life problems data is not publicly available or problem examples are too simple. A practical approach that overcomes this difficulty and has proved very successful for benchmarking standard solving algorithms is to use randomly generated CSPs. This is the approach we consider in this paper to evaluate empirically the proposed algorithms. We extend a random standard CSP generation model [11] to produce random activity constraints, and use the model to implement a random conditional CSP generator. We generate large and diverse problem populations to conduct experimental studies that time algorithm execution, and