Heuristics and look-ahead integration to solve constraint satisfaction problems efficiently

Francisco Menezes* and Pedro Barahona*
Departamento de Informática, Universidade Nova de Lisboa, 2825 Monte da Caparica, Portugal
*Email: fm@fct.unl.pt; Email: pb@fct.unl.pt.

Logic programming languages, such as PROLOG, allow a declarative specification of Constraint Satisfaction Problems (CSPs), freeing the user from specifying more or less complex control directives. However, the price to pay for such flexibility is a loss of efficiency, which makes Logic Programming inadequate to solve CSPs of even moderate size and complexity. To extend the range of applicability of logic programming, several improvements have been proposed. The use of heuristics is one such improvement. Although this usually forces the user to specify some form of control (thus abandoning the pure declarative nature of a logic program), these specifications can be made declarative by making use of some appropriate meta-predicates. Another extension to logic programming that improves its efficiency, is the active use of constraints, as done in the various formulations of constraint logic programming languages. In particular, the use of finite domains is quite adequate to implement look-ahead schemes to efficiently solve several types of CSPs. In this paper, we discuss the complementary nature of heuristics and look-ahead schemes and present a constraint logic programming framework that integrates both these techniques. Results obtained with a time-tabling problem executed on a prototype that implements such a framework are presented, and show that significant efficiency improvements can be achieved when compared with the separate use of the two techniques.

1. Introduction

A CSP consists of a set of variables (the problem's arguments) ranging over specific finite domains and related among themselves by a set of constraints. A solution to a CSP is a set of value assignments for the variables that satisfy all the constraints.

CSPs are usually intractable in that they involve a highly combinatorial search space. More formally, a problem may be considered intractable when there is no polynomial time algorithm to solve it in a deterministic (Turing) machine. However, many instances of such intractable problems, namely the NP-complete [4],
may be solved in polynomial time with the help of a non-deterministic machine, i.e. one with the ability of guessing the adequate values to be assigned to the problem variables.

Logic programming languages, such as PROLOG, allow the declarative specification of CSP problems, where the programmer only has to specify the problem constraints, i.e. its logic, but not its control. Hence, explicit instructions specifying the order in which variables are assigned and values are checked for its consistency need not be specified. Non-determinism in such languages is simulated by a tree search backtracking algorithm, implemented by the interpreter (or compiler) of the language. An example of this behavior is PROLOG, and its backtracking mechanism.

However, the general nature of the language and its tree search algorithm is the cause of several inefficiencies in solving CSPs. Some of this inefficiency can be avoided by the use of heuristics, as advocated in much of the early Artificial Intelligence literature [9,10]. Although this usually forces the user to specify some form of control (thus abandoning the pure declarative nature of a logic program), these specifications can be made declarative by the use of some appropriate meta-predicates.

Another more recent technique to improve the efficiency of logic programming is to replace general unification by more efficient constraint solving in some appropriate domains. In particular, this has been proposed in the design of languages such as CLP [6,7] (for the domain of real numbers), PROLOG-III [2] (for the domain of rational numbers), and CHIP [3] (for finite domains). In all these languages, a form of test and generate paradigm is implemented, in that constraints on variables of the appropriate domains are used actively to reduce the search space a priori.

In particular, several look-ahead schemes [8] that have been used to solve CSPs may be quite efficiently implemented in CHIP, making use of the finite domains that are defined in the language [5]. However, to the best of our knowledge, the dynamic definition of variable domains and the full access to these domains have not been explored thoroughly. As a matter of fact, the designers of the language usually emphasize the use of look-ahead schemes alone, paying little attention to its integration with heuristics.

In this paper, we discuss the use of heuristics and look-ahead schemes, and show that these techniques do not exclude each other, being rather complementary. In fact, whereas look-ahead schemes reduce the search space, the use of heuristics enables a more efficient search through this (reduced) search space. As such, this paper is a case of the integration of the two techniques within the same framework.

The structure of the paper is as follows. In section 2, we analyze the use of heuristics and look-ahead schemes to improve the efficiency of CSP solving in logic programming, and show how these techniques may be integrated in a single framework. In section 3, we present a time-tableing problem that is used for performance evaluation purposes. In section 4, we show experimental results obtain with the various approaches previously presented in section 2. Finally, the main conclusions are summarized in section 5.