Constraint Processing

Roman Barták
Charles University in Prague, Czech Republic

INTRODUCTION

Constraints appear in many areas of human endeavour starting from puzzles like crosswords (the words can only overlap at the same letter) and recently popular Sudoku (no number appears twice in a row) through everyday problems such as planning a meeting (the meeting room must accommodate all participants) till solving hard optimization problems for example in manufacturing scheduling (a job must finish before another job). Though all these problems look like being from completely different worlds, they all share a similar base – the task is to find values of decision variables, such as the start time of the job or the position of the number at a board, respecting given constraints. This problem is called a Constraint Satisfaction Problem (CSP).

Constraint processing emerged from AI research in 1970s (Montanary, 1974) when problems such as scene labelling were studied (Waltz, 1975). The goal of scene labelling was to recognize a type of line (and then a type of object) in the 2D picture of a 3D scene. The possible types were convex, concave, and occluding lines and the combination of types was restricted at junctions of lines to be physically feasible. This scene labelling problem is probably the first problem formalised as a CSP and some techniques developed for solving this problem, namely arc consistency, are still in the core of constraint processing. Systematic use of constraints in programming systems has started in 1980s when researchers identified a similarity between unification in logic programming and constraint satisfaction (Gallaire, 1985) (Jaffar & Lassez, 1987). Constraint Logic Programming was born. Today Constraint Programming is a separate subject independent of the underlying programming language, though constraint logic programming still plays a prominent role thanks to natural integration of constraints into a logic programming framework.

This article presents mainstream techniques for solving constraint satisfaction problems. These techniques stay behind the existing constraint solvers and their understanding is important to exploit fully the available technology.

BACKGROUND

Constraint Satisfaction Problem is formally defined as a triple: a finite set of decision variables, a domain of possible values, and a finite set of constraints restricting possible combinations of values to be assigned to variables. Although the domain can be infinite, for example real numbers, frequently, a finite domain is assumed. Without lost of generality, the finite domain can be mapped to a set of integers which is the usual case in constraint solvers. This article covers finite domains only. In many problems, each variable has its own domain which is a subset of the domain from the problem definition. Such domain can be formally defined by a unary constraint. We already mentioned that constraints restrict possible combinations of values that the decision variables can take. Typically, the constraint is defined over a subset of variables, its scope, and it is specified either extensionally, as a set of value tuples satisfying the constraint, or intentionally, using a logical or arithmetical formula. This formula, for example $A < B$, then describes which value tuples satisfy the constraint. A small example of a CSP is ({A, B, C}, {1, 2, 3}, {A < B, B < C}).

The task of constraint processing is to instantiate each decision variable by a value from the domain in such a way that all constraints are satisfied. This instantiation is called a *feasible assignment*. Clearly, the problem whether there exists a feasible assignment for a CSP is NP-complete – problems like 3SAT or knapsack problem (Garey & Johnson, 1979) can be directly encoded as CSPs. Sometimes, the core constraint satisfaction problem is accompanied by a so called objective function defined over (some) decision variables and we get a *Constrained Optimisation Problem*. Then the task is to select among the feasible assignments the assign-

ment that minimizes (or maximizes) the value of the objective function. This article focuses on techniques for finding a feasible assignment but these techniques can be naturally extended to optimization problems via a well-known branch-and-bound technique (Van Hentenryck, 1989).

There are several comprehensive sources of information about constraint satisfaction starting from journal surveys (Kumar, 1992) (Jaffar & Maher, 1996) through on-line tutorials (Barták, 1998) till several books. Van Hentenryck's book (1989) was a pioneering work showing constraint satisfaction in the context of logic programming. Later Tsang's book (1993) focuses on constraint satisfaction techniques independently of the programming framework and it provides full technical details of most algorithms described later in this article. Recent books cover both theoretical (Apt, 2003) and practical aspects (Marriott & Stuckey, 1998), provide good teaching material (Dechter, 2003) or in-depth surveys of individual topics (Rossi *et al*., 2006). We should not forget about books showing how constraint satisfaction technology is applied in particular areas; scheduling problems play a prominent role here (Baptiste *et al*., 2001) because constraint processing is exceptionally successful in this area.

CONSTRAINT SATISFACTION TECHNIQUES

Constraint satisfaction problems over finite domains are basically combinatorial problems so they can be solved by exploring the space of possible (partial or complete) instantiations of decision variables. Later in this section we will present the typical search algorithms used in constraint processing. However, it should be highlighted that constraint processing is not simple enumeration and we will also show how so called consistency techniques contribute to solving CSPs.

Systematic Search

Search is a core technology of artificial intelligence and many search algorithms have been developed to solve various problems. In case of constraint processing we are searching for a feasible assignment of values to variables where the feasibility is defined by the constraints. This can be done in a backtracking manner where we assign a value to a selected variable and check whether the constraints whose scope is already instantiated are satisfied. In the positive case, we proceed to the next variable. In the negative case, we try another value for the current variable or if there are no more values we backtrack to the last instantiated variable and try alternative values there. The following code shows the skeleton of this procedure called historically *labelling* (Waltz, 1975). Notice that the consistency check may prune domains of individual variables, which will be discussed in the next section.

```
procedure labelling(V,D,C)
        if all variables from V are assigned then return V
        select not-yet assigned variable x from V
        for each value v from D_x do
                (TestOK,D') ← consistent(V,D,C∪{x=v})
                if TestOK=true then
                        R ← labelling(V,D',C)
                         if R ≠ fail then return R
        end for
        return fail
end labelling
```

The above backtracking mechanism is parameterized by variable and value selection heuristics that decide about the order of variables for instantiation and about the order in which the values are tried. While value ordering is usually problem dependent and problem-independent heuristics are not frequently used due to their computational complexity, there are popular problem-independent variable ordering heuristics. Variable ordering is based on a so called *first-fail principle* formulated by Haralick and Eliot (1980) which says that the variable whose instantiation will lead to a failure with the highest probability should be tried first. A typical instance of this principle is a *dom* heuristic which prefers variables with the smallest domain for instantiation. There exist other popular variable ordering heuristics (Rossi *et al*., 2006) such as *dom+deg* or *dom*/*deg*, but their detail description is out scope of this short article.

Though the heuristics influence (positively) efficiency of search they cannot resolve all drawbacks of backtracking. Probably the main drawback is ignoring the information about the reason of constraint infeasibility. If the algorithm discovers that no value can be assigned to a variable, it blindly backtracks to the last instantiated variable though the reason of the conflict may be elsewhere. There exist techniques like backjumping that can detect the variable whose instantiation caused the problem and backtrack (backjump) to this