A DEEP DIVE INTO BACKTRACKING: SOLVING CSPS WITH EFFICIENCY AND ACCURACY

Nguyen Hoang Tan
Department of Computer Science
University of Information Technology
Linh Trung, Thu Duc
21521313@gm.uit.edu.vn

ABSTRACT

Over the past twenty five years many backtracking algorithms have been developed for constraint satisfaction problems. This article describes the basic backtrack search within the search space framework and then presents a number of improvements developed in the past two decades, branching strategies, constraint propagation, nogood recording, backjumping, heuristics for variable and value ordering, randomization and restart strategies, and alternatives to depth-first search.

1 Introduction

There are three main algorithmic techniques for solving constraint satisfaction problems: backtracking search, local search, and dynamic programming. In this article, I survey backtracking search algorithms.

An algorithm for solving a constraint satisfaction problem (CSP) can be either complete or incomplete. Complete, or systematic algorithms, come with a guarantee that a solution will be found if one exists, and can be used to show that a CSP does not have a solution and to find a provably optimal solution. Backtracking search algorithms and dynamic programming algorithms are, in general, examples of complete algorithms. Incomplete, or non-systematic algorithms, cannot be used to show a CSP does not have a solution or to find a provably optimal solution. However, such algorithms are often effective at finding a solution if one exists and can be used to find an approximation to an optimal solution. Local or stochastic search algorithms are examples of incomplete algorithms.

Of the two classes of algorithms that are complete—backtracking search and dynamic programming—backtracking search algorithms are currently the most important in practice. The drawbacks of dynamic programming approaches are that they often require an exponential amount of time and space, and they do unnecessary work by finding, or making it possible to easily generate, all solutions to a CSP. However, one rarely wishes to find all solutions to a CSP in practice. In contrast, backtracking search algorithms work on only one solution at a time and thus need only a polynomial amount of space.

Since the first formal statements of backtracking algorithms over 40 years ago, many techniques for improving the efficiency of a backtracking search algorithm have been suggested and evaluated. In this article, I survey some of the most important techniques including branching strategies, constraint propagation, nogood recording, backjumping, heuristics for variable and value ordering, randomization and restart strategies, and alter- natives to depth-first search. The techniques are not always orthogonal and sometimes combining two or more techniques into one algorithm has a multiplicative effect (such as combining restarts with nogood recording) and sometimes it has a degradation effect (such as increased constraint propagation versus backjumping). Given the many possible ways that these techniques can be combined together into one algorithm, I also survey work on comparing backtracking algorithms. The best combinations of these techniques result in robust backtracking algorithms that can now routinely solve large, hard instances that are of practical importance.

2 Preliminaries

In this section, I first define the constraint satisfaction problem followed by a brief review of the needed background on backtracking search.

Definition 2.1 (CSP)**.** *A constraint satisfaction problem (CSP) consists of a set of variables, $X = \{x_1, ..., x_n\}$; a set of values, $D = \{a_1, ..., a_d\}$, where each variable $x_i \in X$ has an associated finite domain $dom(x_i) \subseteq D$ of possible values; and a collection of constraints.*

As a running example in this survey, I will use the 6-queens problem: how can we place 6 queens on a 6×6 chess board so that no two queens attack each other. As one possible CSP model, let there be a variable for each column of the board $\{x_1, ..., x_6\}$, each with domain $dom(x_i) = \{1, ..., 6\}$. Assigning a value j to a variable x_i means placing a queen in row j, column i. Between each pair of variables x_i and x_j , $1 \leq i < j \leq 6$, there is a constraint $C(x_i, x_j)$, given by $(x_i \neq x_j) \wedge (|i-j| \neq |x_i - x_j|)$. One possible solution is given by $\{x_1 = 4, x_2 = 1, x_3 = 5, x_4 = 2, x_5 = 6, x_6 = 3\}$.