Late-Breaking Developments in the Field of Artificial Intelligence
Papers Presented at the Twenty-Seventh AAAI Conference on Artificial Intelligence

Partial Domain Search Tree for Constraint-Satisfaction Problems

Guni Sharon
ISE Department
Ben-Gurion University
Israel
gunisharon@gmail.com

Ariel Felner
ISE Department
Ben-Gurion University
Israel
felner@bgu.ac.il

Roni Stern
ISE Department
Ben-Gurion University
Israel
roni.stern@gmail.com

Nathan Sturtevant
Department of Computer Science
University of Denver
USA
sturtevant@cs.du.edu

Abstract

CSP solvers usually search a partial assignment search tree. We present a new formalization for CSP solvers, which spans a conceptually different search tree, where each node represents subsets of the original domains for the variables. We experiment with a simple backtracking algorithm for this search tree and show that it outperforms a simple backtracking algorithm on the traditional search tree in many cases.

Introduction

In *constraint satisfaction problems* (CSP), we are given a set of variables $Var = \{var_1, var_2 \dots var_n\}$, a set of domains $D = \{d_1, d_2 \dots d_n\}$ (each variable var_i must take a value from its domain d_i), and a set of constraints C. The goal is to assign each variable a value from its domain so that no constraint is violated. Constraints may be *global*, e.g. all assignments may not sum up to a given value, or *local*, i.e. a given set of values may not be assigned to a given set of variables. Binary constraints are the most basic form of local constraints. A binary constraint is a 4-tuple $\{var_i, val_j, var_t, val_k\}$ where the assignment $var_i \leftarrow val_j$ and $var_t \leftarrow val_k$ is illegal. An assignment is *consistent* if it does not violate any of the constraints. A solution is a full assignment which is consistent. When we make an assignment where a constraint $\{var_i, val_j, var_t, val_k\}$ is violated, we say that var_i is in *conflict* with var_t. For simplicity, we only relate in this work to binary CSPs, however the ideas presented here can easily be generalized.

Traditional CSP search tree.

The traditional approach for solving CSP is to search the partial assignment tree. In a *partial assignment* we assign values to a subset of the variables. The assignment must be consistent. If all the variables were successfully assigned values (*full assignment*) the assignment can be declared as a solution. The size of the *partial assignment tree* is $O(\prod_i d_i)$.

The basic approach for searching the partial assignment tree is using a depth-first backtracking algorithm in the *partial assignment search tree* (PAST). In a given order, each variable var_i is assigned a value from its domain d_i that is consistent with all previously assigned variables. If no such value exists the algorithm backtracks and re-assigns the previous variable. If the last variable was successfully assigned, the search halts and the current assignment is returned as a solution. Many enhancements have been presented to the basic backtracking algorithm. Forward Checking (FC) and Arc Consistency (AC) are forms of lookahead for detecting and avoiding infeasible assignments. Conflict-Based Back Jumping (CBJ) and Dynamic Ordering (DBT) dynamically change the order in which variables are considered. These techniques are possible enhancements to the basic backtracking and they provide significant speed-ups.

The partial-domain search tree

We present a novel search tree, called the *partial-domain search tree* (PDST) for CSP based on our work in the multi-agent pathfinding field (Sharon *et al.* 2012).

Nodes in PDST

Each node in the partial-domain search tree consists of a set of domains $D' = \{d'_1, d'_2 \dots d'_n\}$ where $\forall i : d'_i \subseteq d_i$. In the root node $D' = D$ (i.e., $\forall i : d'_i = d_i$). Unlike the traditional approach (PAST) where each node has a *partial assignment*, in PDST each node consists of a *full* assignment (not necessarily consistent) that is chosen from the current set of domains D'. In other words, every variable var_i is assigned a value from the corresponding d'_i. If the chosen assignment is consistent the state is declared a goal and the solution is returned.

There are several ways this assignment can be chosen, such as choosing the first possible value or randomly. We have found success with the *minimum conflicts heuristic* (MCH) which assigns values greedily according to the minimum number of conflicts with other existing assignments.

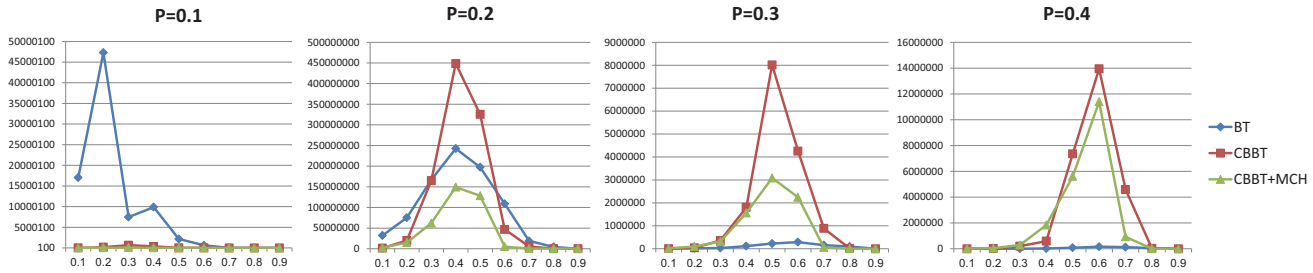

Figure 1: Checks vs. q value for different p values.

Generating successors

If a node n in PDST is not a goal node, meaning that its assignment has at least one internal conflict, it generates successors in order to *resolve* this conflict. In order to guarantee completeness, we generate successors with all possible ways to resolve this conflict. Assume that in node n var_i takes val_j (i.e., $var_i \leftarrow val_j$), we define the following operators:

Block(var_i, val_j)**:** Remove val_j from d'_i. This operator blocks var_i from being assigned with val_j in subsequent assignments.

Lock(var_i, val_j)**:** Remove all values from d'_i except for val_j. This operator *locks* var_i to only take val_j in subsequent assignments.

Assume that node n has a conflict of $\{var_1, val_1, var_2, val_2\}$. In order to resolve this conflict, at least one of the two variables should be blocked from taking the conflicting value in subsequent assignments. There are three possible ways to do this and thus node n has three successors: **(1)** Lock var_2 to only take val_2 and block var_1 from taking val_1. **(2)** Lock var_1 to only take val_1 and block var_2 from taking val_2. **(3)** Block both var_1 and var_2 from taking val_1 and val_2, respectively. Completeness is guaranteed as any valid solution must fall in one (and only one) of the branches. This also prevents duplicates.

Example problem

Figure 2 presents the root R and its three successors for a CSP problem with three variables (v_1, $_2$ and $_3$) each with a domain $d_i = \{1, 2, 3\}$. At R, $\forall i : d'_i = d_i$ (upper left). Without loss of generality, The assignment in R is $\forall i : v_i \leftarrow 1$ (upper right)). Assume that this assignment is not consistent due to the following conflict $\{v_1, 1, v_2, 1\}$ (bottom). R is declared as non-goal and its successors are generated. At the right successor v_1 is blocked from taking the value 1. and v_2 is locked to $v \leftarrow 1$ and so $d'_1 = \{2, 3\}$ and $d'_2 = \{1\}$. At the middle successor v_1 is locked to take 1 and v_2 is blocked from taking 1. At the left successor both v_1 and v_2 are blocked from taking 1. If the new nodes are also non-goal the process continues and each one of them branches three ways. In fact, one can search this tree in any possible manner, until it reaches a goal node.

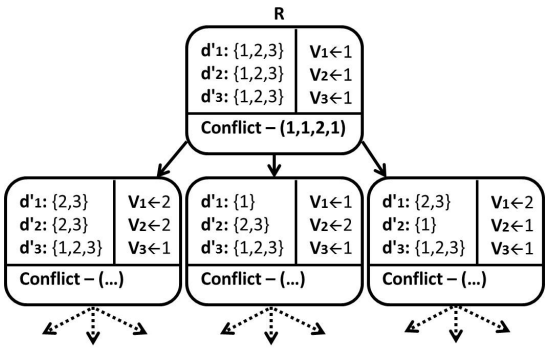

Figure 2: An example of a partial domain state tree.

Experimental results

There are 2^{d_i} possible partial domains for variable i and in total $\prod_i 2^{d_i}$ *partial domains* for all variables. This is larger than $\prod_i d_i$ - the size of the traditional *partial assignment* tree. However, finding a solution can be faster in PDST as more states can lead to a solution.

We experimented with three algorithms. 1) Simple backtracking on PAST (labeled BT) 2) simple backtracking on the PDST (called *conflict-based backtracking* labeled CBBT) 3) simple backtracking on PDST with the minimum conflicts heuristic (labeled CBBT+MCH). Let p be the probability that any two variables contain conflicting values (tightness). For each conflicting variables var_1, var_2 Let q be the probability that an arbitrary assignment $\{var_1, val_i, var_2, val_j\}$ belong to the constraint list (density).

The y-axis in Figure 1 presents the average number of conflict checks CC (a common measure in CSP literature (Balafoutis *et al.* 2011)) each algorithm performed. Each frame is for a different p-value from $\{0.1, 0.2, 0.3, 0.4\}$ while the x-axis is the q-values. There were 15 variables and $\forall i : |d_i| = 5$.

All curves show an *easy-hard-easy* behavior as q increases; a known phenomenon for CSP problems. For $p = 0.1$ CBBT and CBBT+MCH clearly dominate BT. For $p = 0.2$ CBBT+MCH is clearly the winner while CBBT is better than BT on the easy problems. However, for larger values of p BT performs less checks than the CBBT variants. Higher p values means more constraints. This causes more conflicts to be encountered and leads to a blowup in the PDST. Using the MCH heuristic incurs overhead, but results in a smaller number of checks.

Discussion and future work

From our initial results, CBBT outperforms BT on problems with small p. We believe that this is a promising novel direction to further pursue. Our research will focus in four main directions: **(1)** Laying theoretical foundation and mathematical analysis for the PDST approach. **(2)** Using

arc-consistency methods such as AC-4 (Mohr and Henderson 1986) to speedup CBBT. **(3)** Incorporating known fast greedy algorithms such as local search to the CBBT framework as methods for variables assignment in each PDST node. This will increase the chances of finding a solution in early stages of the search while still preserving completeness. **(4)** Compare new versions of CBBT (as suggested in 2 and 3) to state-of-the-art CSP solvers.

References

Thanasis Balafoutis, Anastasia Paparrizou, Kostas Stergiou, and Toby Walsh. New algorithms for max restricted path consistency. *Constraints*, 16(4):372–406, 2011.

Roger Mohr and Thomas C. Henderson. Arc and path consistency revisited. *Artif. Intell.*, 28(2):225–233, 1986.

Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent path finding. In *AAAI*, 2012.