Improvements for Backtracking Algorithms

Pyry Koivisto

March 4, 2009

Outline

So what's this all about?

In this presentation on Constraint Satisfaction Problems (CSPs), we continue examining backtracking algorithms.
More specifically we examine techniques for improving the efficiency of backtracking algorithms. We will assume that some form of constraint propagation is performed.

Performance-critical decisions

There are several decisions one must make in designing a backtracking algorithm that may greatly affect its performance:

Performance-critical decisions

There are several decisions one must make in designing a backtracking algorithm that may greatly affect its performance:

- how to choose which variable to branch on?

Performance-critical decisions

There are several decisions one must make in designing a backtracking algorithm that may greatly affect its performance:

- how to choose which variable to branch on?
- in which order to assign values for a variable?

Performance-critical decisions

There are several decisions one must make in designing a backtracking algorithm that may greatly affect its performance:

- how to choose which variable to branch on?
- in which order to assign values for a variable?
- when to give up on an unpromising branch of the search tree?

Performance-critical decisions

There are several decisions one must make in designing a backtracking algorithm that may greatly affect its performance:

- how to choose which variable to branch on?
- in which order to assign values for a variable?
- when to give up on an unpromising branch of the search tree?
- ..and more.

Outline

Generally speaking

There are a couple of fundamental decisions to make when using a backtracking search to solve a CSP – which variable to branch on and how to choose which value to give to a variable.
Heuristics for these questions are referred to as variable and value ordering heuristics.

Some definitions

- We call an ordering static if the ordering is fixed prior to search and dynamic if it is determined as the search chugs along.
- We say that an ordering is optimal if the ordering guides the search to a solution (or conclusion that there exists none) with a minimum number of nodes visited.
- Finally, heuristics can be application-independent or application-dependent. In this presentation, we focus on the former ones.

Outline

Variable Ordering Heuristics

Consider a situation where the backtracking search should extend a node p. Variable ordering heuristics attempt to choose the next variable x to branch on.
Most variable ordering heuristics can be grouped into two categories.

- heuristics based on domain sizes
- heuristics based on the structure of the CSP

Variable ordering: domain sizes

Definition

Let $rem(x|p)$ be the number of values remaining in $dom(x)$ after constraint propagation, given a set of branching constraints p. We define **dom** to be the heuristic of choosing the unassigned variable x that minimizes $rem(x|p)$.

dom in conjunction with FC has been shown to be an effective combination.

Variable ordering: domain sizes

dom has been studied a lot and much effort has also gone into improving it. One commonly used variant is **dom+deg**.

Definition

Let the degree of an unassigned variable x be the number of constraints involving x and at least one other unassigned variable. **dom+deg** is the heuristic of choosing the variable that minimizes $rem(x|p)$ and breaks ties by choosing the variable with the highest degree.

Variable ordering: structures

Consider the *constraint graph*: a vertex for each variable in the CSP and edges between vertices x and y if there exists a constraint C such that $x, y \in \mathit{vars}(C)$.
Structure-guided variable ordering heuristics take advantage of structures in constraint graphs that real-world problems often contain. These heuristics can in some cases be used to bound the worst-case of a backtracking algorithm.

Variable ordering: structures

A variable ordering heuristic based on structures proposed by Dechter and Pearl is to first instantiate variables that cut cycles in the constraint graph.
Once all cycles have been cut, the graph is a tree and can be solved quickly using arc consistency.

Outline

Value Ordering Heuristics

Suppose that a variable ordering heuristic has chosen variable x to be branched on next. The task of a value ordering heuristic is to choose the next value $a \in dom(x)$. Usually value ordering heuristics endeavour to choose the next value that is most likely to succeed or be a part of a solution.

One heuristic due to Ginsberg et al. is to choose the value $a \in dom(x)$ that maximizes $\prod_y rem(y|p \cup \{x = a\})$, where y runs over all the unassigned variables.

Outline

Why Randomization and Restarts?

Fairly small changes to ordering heuristics may cause substantial differences in running times of a backtracking algorithm; this is due to mistakes (unoptimal choices) made by the heuristics.
A technique called randomization and restarts can be quite effective in countering these differences.

Why Randomization and Restarts?

Fairly small changes to ordering heuristics may cause substantial differences in running times of a backtracking algorithm; this is due to mistakes (unoptimal choices) made by the heuristics.
A technique called randomization and restarts can be quite effective in countering these differences.

- The basic idea: if the backtracking algorithm has backtracked from a deadend "far enough", the backtracking algorithm is restarted with different orderings.

Outline

Randomization

Randomization entails modifying the heuristics for ordering variables or values; both approaches have been recommended. Gomes et al. propose randomizing the variable ordering by

- randomized tie breaking or
- ranking the variables and choosing from a set of variables within a small factor of the best variable.

However one implements randomization, it should provide enough alternatives at the top of the search tree.

Outline

Restart Strategies

Definition

A restart strategy $S = (t_1, t_2, t_3, \ldots)$ is an infinite sequence of steps $t_i \in \mathbb{Z}^+$. A *fixed cutoff* strategy is one where all the t_i are equal.

We interpret this as a schedule where the randomized backtracking algorithm is run for t_1 steps. If no solution is found within that cutoff, the algorithm is run for t_2 steps etc.

Of course, to implement a restarting strategy, one must then decide what counts as a step in the computation. One might for example restart after the number of backtracks exceeds some cutoff.

Restart Strategies Cont'd

Some suggested restart strategies:

- an optimal strategy $S_{t^*} = (t^*, t^*, t^*, \ldots)$ – given full knowledge of the runtime distribution, supposedly optimal for some fixed cutoff t^*. In practice, the runtime distribution is of course not known.
- a geometric strategy $S_g = (1, r, r^2, \ldots)$ with $1 < r < 2$, proposed by Walsh – has the advantage of the cutoff values increasing relatively quickly and no need to find the optimal cutoff like in a fixed cutoff strategy

Outline

Runtime distributions for which restarts are useful

As mentioned, even small changes to problem instances or the search algorithm can lead to wildly varying runtimes.
This unpredictability in running times can often be explained by heavy-tailed runtime distributions. These are distributions where the tail probability decays polynomially, ie. there is a significant probability that the backtracking algorithm will run for a long time.

Outline

Optimization CSPs

Finding a solution to a CSP is not always enough, there are application areas of constraint programming (eg. scheduling) where one must also optimize/minimize an objective function f. We add to the CSP an *objective constraint* $c = f(X)$, where X is the set of variables.

Optimization CSPs

Solving optimization CSPs is usually approached as solving a sequence of CSPs. One suggested approach is a constrained-based version of branch-and-bound. In this approach backtracking search is performed to find some solution satisfying the original constraints and if a solution is found, we add an objective constraint $c < f(S)$ that only admits solutions better than the current one. When we can no longer decrease the objective constraint, the last solution has been proven optimal.

Outline

Summary

- Ordering heuristics can have a great effect on the performance of backtracking algorithms
- Randomization and restart strategies are nowadays very commonly used to benefit from the variance in runtimes of different heuristics