Journal of Artificial Intelligence Research 1 (1993) 25-46 Submitted 7/93; published 8/93

# Dynamic Backtracking

**Matthew L. Ginsberg** GINSBERG@CS.UOREGON.EDU
*CIRL, University of Oregon,*
*Eugene, OR 97403-1269 USA*

## Abstract

Because of their occasional need to return to shallow points in a search tree, existing backtracking methods can sometimes erase meaningful progress toward solving a search problem. In this paper, we present a method by which backtrack points can be moved deeper in the search space, thereby avoiding this difficulty. The technique developed is a variant of dependency-directed backtracking that uses only polynomial space while still providing useful control information and retaining the completeness guarantees provided by earlier approaches.

## 1. Introduction

Imagine that you are trying to solve some constraint-satisfaction problem, or CSP. In the interests of definiteness, I will suppose that the CSP in question involves coloring a map of the United States subject to the restriction that adjacent states be colored differently.

Imagine we begin by coloring the states along the Mississippi, thereby splitting the remaining problem in two. We now begin to color the states in the western half of the country, coloring perhaps half a dozen of them before deciding that we are likely to be able to color the rest. Suppose also that the last state colored was Arizona.

At this point, we change our focus to the eastern half of the country. After all, if we can't color the eastern half because of our coloring choices for the states along the Mississippi, there is no point in wasting time completing the coloring of the western states.

We successfully color the eastern states and then return to the west. Unfortunately, we color New Mexico and Utah and then get stuck, unable to color (say) Nevada. What's more, backtracking doesn't help, at least in the sense that changing the colors for New Mexico and Utah alone does not allow us to proceed farther. Depth-first search would now have us backtrack to the eastern states, trying a new color for (say) New York in the vain hope that this would solve our problems out West.

This is obviously pointless; the blockade along the Mississippi makes it impossible for New York to have any impact on our attempt to color Nevada or other western states. What's more, we are likely to examine every *possible* coloring of the eastern states before addressing the problem that is actually the source of our difficulties.

The solutions that have been proposed to this involve finding ways to backtrack directly to some state that might actually allow us to make progress, in this case Arizona or earlier. Dependency-directed backtracking (Stallman & Sussman, 1977) involves a direct backtrack to the source of the difficulty; backjumping (Gaschnig, 1979) avoids the computational overhead of this technique by using syntactic methods to estimate the point to which backtrack is necessary.

In both cases, however, note that although we backtrack to the source of the problem, we backtrack *over* our successful solution to half of the original problem, discarding our solution to the problem of coloring the states in the East. And once again, the problem is worse than this – after we recolor Arizona, we are in danger of solving the East yet again before realizing that our new choice for Arizona needs to be changed after all. We won't examine every possible coloring of the eastern states, but we are in danger of rediscovering our successful coloring an exponential number of times.

This hardly seems sensible; a human problem solver working on this problem would simply ignore the East if possible, returning directly to Arizona and proceeding. Only if the states along the Mississippi needed new colors would the East be reconsidered – and even then only if no new coloring could be found for the Mississippi that was consistent with the eastern solution.

In this paper we formalize this technique, presenting a modification to conventional search techniques that is capable of backtracking not only to the most recently expanded node, but also directly to a node elsewhere in the search tree. Because of the dynamic way in which the search is structured, we refer to this technique as *dynamic backtracking*.

A more specific outline is as follows: We begin in the next section by introducing a variety of notational conventions that allow us to cast both existing work and our new ideas in a uniform computational setting. Section 3 discusses backjumping, an intermediate between simple chronological backtracking and our ideas, which are themselves presented in Section 4. An example of the dynamic backtracking algorithm in use appears in Section 5 and an experimental analysis of the technique in Section 6. A summary of our results and suggestions for future work are in Section 7. All proofs have been deferred to an appendix in the interests of continuity of exposition.

## 2. Preliminaries

**Definition 2.1** *By a* constraint satisfaction problem $(I, V, \kappa)$ *we will mean a set $I$ of variables; for each $i \in I$, there is a set $V_i$ of possible values for the variable $i$. $\kappa$ is a set of constraints, each a pair $(J, P)$ where $J = (j_1, \ldots, j_k)$ is an ordered subset of $I$ and $P$ is a subset of $V_{j_1} \times \cdots \times V_{j_k}$.*

*A* solution *to the* CSP *is a set $v_i$ of values for each of the variables in $I$ such that $v_i \in V_i$ for each $i$ and for every constraint $(J, P)$ of the above form in $\kappa$, $(v_{j_1}, \ldots, v_{j_k}) \in P$.*

In the example of the introduction, $I$ is the set of states and $V_i$ is the set of possible colors for the state $i$. For each constraint, the first part of the constraint is a pair of adjacent states and the second part is a set of allowable color combinations for these states.

Our basic plan in this paper is to present formal versions of the search algorithms described in the introduction, beginning with simple depth-first search and proceeding to backjumping and dynamic backtracking. As a start, we make the following definition of a partial solution to a CSP:

**Definition 2.2** *Let $(I, V, \kappa)$ be a* CSP*. By a* partial solution *to the* CSP *we mean an ordered subset $J \subseteq I$ and an assignment of a value to each variable in $J$.*