Journal of Artificial Intelligence Research 1 (1993) 1-15 Submitted 6/91; published 9/91

Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems

Kelechi Ikegwu IKEGWU2@ILLINOIS.EDU
NASA Ames Research Center, Mail Stop: 244-7,
Moffett Field, CA 94035 USA

Micheal Hao ???@ILLINOIS,.EDU
Space Telescope Science Institute, 3700 San Martin Drive,
Baltimore, MD 21218 USA

Philip Laird LAIRD@PTOLEMY.ARC.NASA.GOV
NASA Ames Research Center, AI Research Branch, Mail Stop: 269-2,
Moffett Field, CA 94035 USA

Abstract

This paper describes a simple heuristic approach to solving large-scale constraint satisfaction and scheduling problems. In this approach one starts with an inconsistent assignment for a set of variables and searches through the space of possible repairs. The search can be guided by a value-ordering heuristic, the *min-conflicts heuristic*, that attempts to minimize the number of constraint violations after each step. The heuristic can be used with a variety of different search strategies. We demonstrate empirically that on the n-queens problem, a technique based on this approach performs orders of magnitude better than traditional backtracking techniques. We also describe a scheduling application where the approach has been used successfully. A theoretical analysis is presented both to explain why this method works well on certain types of problems and to predict when it is likely to be most effective.

1. Introduction

Introduction Stuff

2. RelatedWorks

Related Works Stuff

3. Why does the GDS Network Perform So Well?

Our analysis of the GDS network was motivated by the following question: "Why does the network perform so much better than traditional backtracking methods on certain tasks"? In particular, we were intrigued by the results on the n-queens problem, since this problem has received considerable attention from previous researchers. For n-queens, Adorf and Johnston found empirically that the network requires a linear number of transitions to converge. Since each transition requires linear time, the expected (empirical) time for

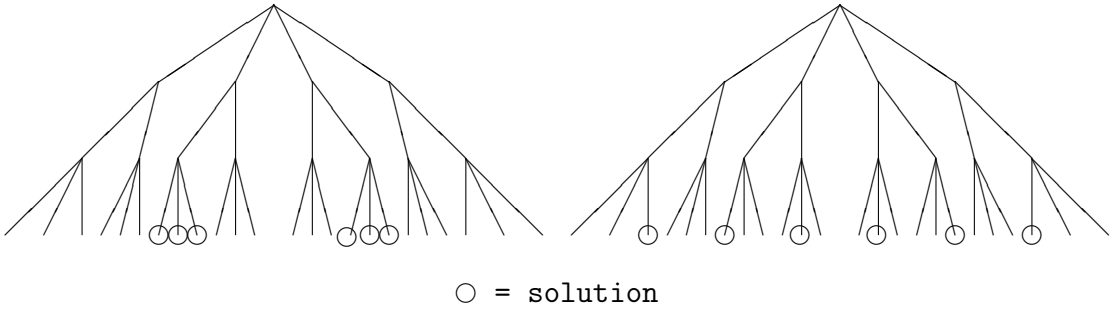

Figure 1: Solutions Clustered vs. Solutions Evenly Distributed

the network to find a solution is $O(n^2)$. To check this behavior, Johnston and Adorf ran experiments with n as high as 1024, at which point memory limitations became a problem.[1]

3.1 Nonsystematic Search Hypothesis

Initially, we hypothesized that the network's advantage came from the nonsystematic nature of its search, as compared to the systematic organization inherent in depth-first backtracking. There are two potential problems associated with systematic depth-first search. First, the search space may be organized in such a way that poorer choices are explored first at each branch point. For instance, in the n-queens problem, depth-first search tends to find a solution more quickly when the first queen is placed in the center of the first row rather than in the corner; apparently this occurs because there are more solutions with the queen in the center than with the queen in the corner (?). Nevertheless, most naive algorithms tend to start in the corner simply because humans find it more natural to program that way. However, this fact by itself does not explain why nonsystematic search would work so well for n-queens. A backtracking program that randomly orders rows (and columns within rows) performs much better than the naive method, but still performs poorly relative to the GDS network.

The second potential problem with depth-first search is more significant and more subtle. As illustrated by figure 1, a depth-first search can be a disadvantage when solutions are not evenly distributed throughout the search space. In the tree at the left of the figure, the solutions are clustered together. In the tree on the right, the solutions are more evenly distributed. Thus, the average distance between solutions is greater in the left tree. In a depth-first search, the average time to find the first solution increases with the average distance between solutions. Consequently depth-first search performs relatively poorly in a tree where the solutions are clustered, such as that on the left (?, ?). In comparison, a search strategy which examines the leaves of the tree in random order is unaffected by solution clustering.

1. The network, which is programmed in Lisp, requires approximately 11 minutes to solve the 1024 queens problem on a TI Explorer II. For larger problems, memory becomes a limiting factor because the network requires approximately $O(n^2)$ space. (Although the number of connections is actually $O(n^3)$, some connections are computed dynamically rather than stored).

We investigated whether this phenomenon explained the relatively poor performance of depth-first search on n-queens by experimenting with a randomized search algorithm, called a Las Vegas algorithm (?). The algorithm begins by selecting a path from the root to a leaf. To select a path, the algorithm starts at the root node and chooses one of its children with equal probability. This process continues recursively until a leaf is encountered. If the leaf is a solution the algorithm terminates, if not, it starts over again at the root and selects a path. The same path may be examined more than once, since no memory is maintained between successive trials.

The Las Vegas algorithm does, in fact, perform better than simple depth-first search on n-queens (?). However, the performance of the Las Vegas algorithm is still not nearly as good as that of the GDS network, and so we concluded that the systematicity hypothesis alone cannot explain the network's behavior.

3.2 Informedness Hypothesis

Our second hypothesis was that the network's search process uses information about the current assignment that is not available to a constructive backtracking program. 's use of an iterative improvement strategy guides the search in a way that is not possible with a standard backtracking algorithm. We now believe this hypothesis is correct, in that it explains why the network works so well. In particular, the key to the network's performance appears to be that state transitions are made so as to reduce the number of outstanding inconsistencies in the network; specifically, each state transition involves flipping the neuron whose output is most inconsistent with its current input. From a constraint satisfaction perspective, it is as if the network reassigns a value for a variable by choosing the value that violates the fewest constraints. This idea is captured by the following heuristic:

> **Min-Conflicts heuristic:**
> *Given:* A set of variables, a set of binary constraints, and an assignment specifying a value for each variable. Two variables *conflict* if their values violate a constraint.
> *Procedure:* Select a variable that is in conflict, and assign it a value that minimizes the number of conflicts. (Break ties randomly.)

We have found that the network's behavior can be approximated by a symbolic system that uses the min-conflicts heuristic for hill climbing. The hill-climbing system starts with an initial assignment generated in a preprocessing phase. At each choice point, the heuristic chooses a variable that is currently in conflict and reassigns its value, until a solution is found. The system thus searches the space of possible assignments, favoring assignments with fewer total conflicts. Of course, the hill-climbing system can become "stuck" in a local maximum, in the same way that the network may become "stuck" in a local minimum. In the next section we present empirical evidence to support our claim that the min-conflicts approach can account for the network's effectiveness.

There are two aspects of the min-conflicts hill-climbing method that distinguish it from standard CSP algorithms. First, instead of incrementally constructing a consistent partial assignment, the min-conflicts method *repairs* a complete but inconsistent assignment by reducing inconsistencies. Thus, it uses information about the current assignment to guide its search that is not available to a standard backtracking algorithm. Second, the use of

```
Procedure INFORMED-BACKTRACK (VARS-LEFT VARS-DONE)
 If all variables are consistent, then solution found, STOP.
 Let VAR = a variable in VARS-LEFT that is in conflict.
 Remove VAR from VARS-LEFT.
 Push VAR onto VARS-DONE.
 Let VALUES = list of possible values for VAR in ascending order according
              to number of conflicts with variables in VARS-LEFT.
 For each VALUE in VALUES, until solution found:
   If VALUE does not conflict with any variable that is in VARS-DONE,
   then Assign VALUE to VAR.
        Call INFORMED-BACKTRACK(VARS-LEFT VARS-DONE)
   end if
 end for
end procedure

Begin program
 Let VARS-LEFT = list of all variables, each assigned an initial value.
 Let VARS-DONE = nil
 Call INFORMED-BACKTRACK(VARS-LEFT VARS-DONE)
End program
```

Figure 2: Informed Backtracking Using the Min-Conflicts Heuristic

a hill-climbing strategy rather than a backtracking strategy produces a different style of search.

3.2.1 Repair-Based Search Strategies

(This is a example of a third level section.) Extracting the method from the network enables us to tease apart and experiment with its different components. In particular, the idea of repairing an inconsistent assignment can be used with a variety of different search strategies in addition to hill climbing. For example, we can backtrack through the space of possible repairs, rather than using a hill-climbing strategy, as follows. Given an initial assignment generated in a preprocessing phase, we can employ the min-conflicts heuristic to order the choice of variables and values to consider, as described in figure 2. Initially, the variables are all on a list of VARS-LEFT, and as they are repaired, they are pushed onto a list of VARS-DONE. The algorithm attempts to find a sequence of repairs, such that no variable is repaired more than once. If there is no way to repair a variable in VARS-LEFT without violating a previously repaired variable (a variable in VARS-DONE), the algorithm backtracks.

Notice that this algorithm is simply a standard backtracking algorithm augmented with the min-conflicts heuristic to order its choice of which variable and value to attend to. This illustrates an important point. The backtracking repair algorithm incrementally extends a consistent partial assignment (i.e., VARS-DONE), as does a constructive backtracking program, but in addition, uses information from the initial assignment (i.e., VARS-LEFT) to

bias its search. Thus, it is a type of *informed backtracking*. We still characterize it as repair-based method since its search is guided by a complete, inconsistent assignment.

4. Experimental Results

[section ommitted]

5. A Theoretical Model

[section ommitted]

6. Discussion

[section ommitted]

Acknowledgments

The authors wish to thank Hans-Martin Adorf, Don Rosenthal, Richard Franier, Peter Cheeseman and Monte Zweben for their assistance and advice. We also thank Ron Musick and our anonymous reviewers for their comments. The Space Telescope Science Institute is operated by the Association of Universities for Research in Astronomy for NASA.

Appendix A. Probability Distributions for N-Queens

[section ommitted]