Journal of Artificial Intelligence Research 1 (1993) 25-32 Submitted 6/91; published 9/91

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

Steven Minton MINTON@PTOLEMY.ARC.NASA.GOV
Andy Philips PHILIPS@PTOLEMY.ARC.NASA.GOV
Sterling Federal Systems, AI Research Branch, Mail Stop: 269-2,
NASA Ames Research Center, Moffett Field, CA 94035, U.S.A.

Mark D. Johnston JOHNSTON@STSCI.EDU
Space Telescope Science Institute, 3700 San Martin Drive,
Baltimore, MD 21218, U.S.A.

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

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

One of the most promising general approaches for solving combinatorial search problems is to generate an initial, suboptimal solution and then to apply local *repair* heuristics. Techniques based on this approach have met with empirical success on many combinatorial problems, including the traveling salesman and graph partitioning problems (Johnson, Papadimitrou, & Yannakakis, 1988). Such techniques also have a long tradition in AI, most notably in problem-solving systems that operate by debugging initial solutions (Simmons, 1988; Sussman, 1975). In this paper, we describe how this idea can be extended to constraint satisfaction problems (CSPs) in a natural manner.

Most of the previous work on CSP algorithms has assumed a "constructive" backtracking approach in which a partial assignment to the variables is incrementally extended. In contrast, our method (Minton, Johnston, Philips, & Laird, 1990) creates a complete, but inconsistent assignment and then repairs constraint violations until a consistent assignment is achieved. The method is guided by a simple ordering heuristic for repairing constraint violations: identify a variable that is currently in conflict and select a new value that minimizes the number of outstanding constraint violations.

We present empirical evidence showing that on some standard problems our approach is considerably more efficient than traditional constructive backtracking methods. For example, on the n-queens problem, our methods quickly finds solutions to the one million queens problem. We argue that the reason that repair-based methods can outperform constructive methods is because a complete assignment can be more informative in guiding search than a partial assignment. However, the utility of the extra information is domain dependent. To help clarify the nature of this potential advantage, we present a theoretical analysis that describes how various problem characteristics may affect the performance of the method. This analysis shows, for example, how the "distance" between the current assignment and solution (in terms of the minimum number of repairs that are required) affects the expected utility of the heuristic.

The work described in this paper was inspired by a surprisingly effective neural network developed by Tariff and Johnston (Adopt & Johnston, 1990: Johnston & Tariff, 1989) for scheduling astronomical observations on the Hobble Space Telescope. Our heuristic CSP method was distilled from an analysis of the network. In the process of carrying out the analysis, we discovered that the effectiveness of the network has little to do with its connections implementation. Furthermore, the ideas employed in the network can be implemented very efficiently within a symbolic CSP framework. The symbolic implementation is extremely simple. It also has the advantage that several different search strategies can be employed, although we have found that hill-climbing methods are particularly well-suited for the applications that we have investigated.

We begin the paper with a brief review of Tariff and Johnston's neural network, and then describe our symbolic method for heuristic repair. Following this, we describe empirical results with the n-queens problem, graph-colorability problems and the Hobble Space Telescope scheduling application. Finally, we consider a theoretical model identifying general problem characteristics that influence the performance of the method. We include a second gratuitous citation to ourselves to illustrate a short citation (Minton et al., 1990).

2. Previous Work: The GDS Network

By almost any measure, the Hubble Space Telescope scheduling problem is a complex task (Johnston, 1987; Airdrop, 1989). Between ten thousand and thirty thousand astronomical observations per year must be scheduled, subject to a great variety of constraints including power restrictions, observation priorities, time-dependent orbital characteristics, movement of astronomical bodies, stray light sources, etc. Because the telescope is an extremely valuable resource with a limited lifetime, efficient scheduling is a critical concern. An initial scheduling system, developed using traditional programming methods, highlighted the difficulty of the problem; it was estimated that it would take over three weeks for the system to schedule one week of observations. As described in section 4, this problem was remedied by dividing the problem into a long-term scheduling problem and a short the development of a successful constraint-based system to augment the initial system. The constraint-based system produces a high-level schedule and the original system then derives a detailed schedule. A more successful constraint-based system was then developed to augment the original system. At the heart of the constraint-based system is a neural network developed by Adorf and Johnston, the Guarded Discrete Stochastic (GDS) network, which searches for a schedule (Adorf & Johnston, 1990: Johnston & Adorf, 1989).

From a computational point of view the network is interesting because Adorf and Johnston found that it performs well on a variety of tasks, in addition to the space telescope scheduling problem. For example, the network performs significantly better on the n-queens problem than methods that were previously developed. The n-queens problem requires placing n queens on an $n \times n$ chessboard so that no two queens share a row, column or diagonal. The network has been used to solve problems of up to 1024 queens, whereas most heuristic backtracking methods encounter difficulties with problems one-tenth that size (Stone & Stone, 1987).

The GDS network is a modified Hopfield network (Hopfield, 1982). In a standard Hopfield network, all connections between neurons are symmetric. In the GDS network, the main network is coupled asymmetrically to an auxiliary network of guard neurons which restricts the configurations that the network can assume. This modification enables the network to rapidly find a solution for many problems, even when the network is simulated on a serial machine. Unfortunately, convergence to a stable configuration is no longer guaranteed. Thus the network can fall into a local minimum involving a group of unstable states among which it will oscillate. In practice, however, if the network fails to converge after some number of neuron state transitions, it can simply be stopped and started over.

To illustrate the network architecture and updating scheme, let us consider how the network is used to solve binary constraint satisfaction problems. A problem consists of n variables, $X_1 \ldots X_n$, with domains $D_1 \ldots D_n$, and a set of binary constraints. Each constraint $C\alpha(X_i, X_k)$ is a subset of $D_j \times D_k$ specifying incompatible values for a pair of variables. The goal is to find an assignment for each of the variables which satisfies the constraints. (In this paper we only consider the task of finding a single solution, rather than that of finding all solutions.) To solve a CSP using the network, each variable is represented by a separate set of neurons, one neuron for each of the variable's possible values. Each neuron is either “on” or “off”, and in a solution state, every variable will have exactly one of its corresponding neurons “on”, representing the value of that variable. Constraints are represented by inhibitory (i.e., negatively weighted) connections between the neurons. To insure that every variable is assigned a value, there is a guard neuron for each set of neurons representing a variable; if no neuron in the set is on, the guard neuron will provide an excitatory input that is large enough to turn one on. (Because of the way the connection weights are set up, it is unlikely that the guard neuron will turn on more than one neuron.) The network is updated on each cycle by randomly picking a set of neurons that represents a variable, and flipping the state of the neuron in that set whose input is *most inconsistent* with its current output (if any). When all neurons’ states are consistent with their input, a solution is achieved.

To solve the n-queens problem, for example, each of the $n \times n$ board positions is represented by a neuron whose output is either one or zero depending on whether a queen is currently placed in that position or not. (Note that this is a local representation rather than a distributed representation of the board.) If two board positions are inconsistent, then an inhibiting connection exists between the corresponding two neurons. For example, all the neurons in a column will inhibit each other, representing the constraint that two queens cannot be in the same column. For each row, there is a guard neuron connected to each of the neurons in that row which gives the neurons in the row a large excitatory input, enough so that at least one neuron in the row will turn on. The guard neurons thus enforce the constraint that one queen in each row must be on. As described above, the network is updated on each cycle by randomly picking a row and flipping the state of the neuron in that row whose input is most inconsistent with its current output. A solution is realized when the output of every neuron is consistent with its input.

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 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,[1] at which point memory limitations became a problem.

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.

Figure 1: Solutions Clustered vs. Solutions Evenly Distributed

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

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).

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 (Ginsberg & Harvey, 1990; Langley, 1992). In comparison, a search strategy which examines the leaves of the tree in random order is unaffected by solution clustering.

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 (Brassard & Bratley, 1988). 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 (Brassard & Bratley, 1988). In fact, this result was already known. 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.[2] (Break ties randomly.)

We have found that the network's behavior can be approximated by asymbolic 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

2. In general, the heuristic attempts to minimize the number of other variables that will need to be repaired. For binary CSPs, this corresponds to minimizing the number of conflicting variables. For general CSPs, where a single constraint may involve several variables, the exact method of counting the number of variables that will need to be repaired depends on the particular constraint. The space telescope scheduling problem is a general CSP, whereas the other tasks described in this paper are binary CSPs.

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

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 without violating a previously repaired variable (a variable in VARS DONE), the algorithm backtracks.

```
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 ordered 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

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.

3.2.1 Repair-Based Search Strategies

(This is an 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 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.

4. Experimental Results

[Section Omitted]

5. A Theoretical Model

[Section Omitted]

6. Discussion

[Section Omitted]

Acknowledgments

The authors wish to thank Hans-Martin Tariff, Don Rosenthal, Richard Fernier, Peter Cheesman and Monte Zwieback for their assistance and advice. We also thank Ron Music 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

[Appendix Omitted]

References

Interbred, M. (1973). *Cluster analysis for applications*. New York: Academic Press.

Blaming, D., & Hit, E. (1988). Observational learning from internal feedback: A simulation of an adaptive learning method. *Cognitive Science*, *12*, 587–625.

Detrain, R., Janis, A., Steinberg, W., Uplifters, M., Schmidt, J., Sandhi, S., Guppy, K., Lee, S., & Freeholder, V. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disease. *American Journal of Cardiology*, *64*, 304–310.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2*, 139–172.

Gennari, J. H. (1990). *An experimental study of concept formation* (Doctoral dissertation, also Technical Report 90-26). Irvine: University of California, Department of Information and Computer Science.

Gennari, J. H., Langley, P., & Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence*, *40*, 11–61.

Gluck, M., & Corter, J. (1985). Information, uncertainty and the utility of categories. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* (pp. 283–287). Irvine, CA: Lawrence Erlbaum.

Iba, W., & Gennari, J. H. (in press). Learning to recognize movements. In D. Fisher & M. Pazzani (Eds.), *Computational approaches to concept formation*. San Mateo, CA: Morgan Kaufmann.

Michalski, R. S., & Stepp, R. (1983). Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Ends.), *Machine learning: An artificial intelligence approach*. San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Schneider, W., Dames, S. T., & Shivering, R. M. (1984). Automatic and control processing and attention. In R. Parathormone & D. R. Davies (Ends.), *Varieties of attention*. San Diego, CA: Academic Press.