

1 Latex Template for CO600 and CO620 Project
2 Reports

3 Example Name
em123@kent.ac.uk



School of Computing
University of Kent
United Kingdom

Word Count: 6,100

4 May 24, 2018

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

50 expected utility of the heuristic.

51 The work described in this paper was inspired by a surprisingly effec-
52 tive neural network developed by Adorf and Johnston Adorf and Johnston
53 (1990); Johnston and Adorf (1989) for scheduling astronomical observations
54 on the Hubble Space Telescope. Our heuristic CSP method was distilled from
55 an analysis of the network. In the process of carrying out the analysis, we
56 discovered that the effectiveness of the network has little to do with its con-
57 nectionist implementation. Furthermore, the ideas employed in the network
58 can be implemented very efficiently within a symbolic CSP framework. The
59 symbolic implementation is extremely simple. It also has the advantage that
60 several different search strategies can be employed, although we have found
61 that hill-climbing methods are particularly well-suited for the applications
62 that we have investigated.

63 We begin the paper with a brief review of Adorf and Johnston’s neural
64 network, and then describe our symbolic method for heuristic repair. Fol-
65 lowing this, we describe empirical results with the n -queens problem, graph-
66 colorability problems and the Hubble Space Telescope scheduling application.
67 Finally, we consider a theoretical model identifying general problem charac-
68 teristics that influence the performance of the method. We include a second
69 gratuitous citation to ourselves to illustrate a short citation Minton et al.
70 (1990).

71 2 Previous Work: The GDS Network

72 By almost any measure, the Hubble Space Telescope scheduling problem is a
73 complex task Johnston (1987); Waldrop (1989). Between ten thousand and
74 thirty thousand astronomical observations per year must be scheduled, sub-
75 ject to a great variety of constraints including power restrictions, observation
76 priorities, time-dependent orbital characteristics, movement of astronomical
77 bodies, stray light sources, etc. Because the telescope is an extremely valu-
78 able resource with a limited lifetime, efficient scheduling is a critical con-

cern. 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 the development of a successful constraint-based system to augment the initial 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 and Johnston (1990); Johnston and 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 and 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.

110 A problem consists of n variables, $X_1 \dots X_n$, with domains $D_1 \dots D_n$, and a
 111 set of binary constraints. Each constraint $C_\alpha(X_j, X_k)$ is a subset of $D_j \times D_k$
 112 specifying incompatible values for a pair of variables. The goal is to find
 113 an assignment for each of the variables which satisfies the constraints. (In
 114 this paper we only consider the task of finding a single solution, rather than
 115 that of finding all solutions.) To solve a CSP using the network, each vari-
 116 able is represented by a separate set of neurons, one neuron for each of the
 117 variable's possible values. Each neuron is either "on" or "off", and in a solu-
 118 tion state, every variable will have exactly one of its corresponding neurons
 119 "on", representing the value of that variable. Constraints are represented by
 120 inhibitory (i.e., negatively weighted) connections between the neurons. To
 121 insure that every variable is assigned a value, there is a guard neuron for
 122 each set of neurons representing a variable; if no neuron in the set is on, the
 123 guard neuron will provide an excitatory input that is large enough to turn
 124 one on. (Because of the way the connection weights are set up, it is unlikely
 125 that the guard neuron will turn on more than one neuron.) The network is
 126 updated on each cycle by randomly picking a set of neurons that represents
 127 a variable, and flipping the state of the neuron in that set whose input is
 128 *most inconsistent* with its current output (if any). When all neurons' states
 129 are consistent with their input, a solution is achieved.

130 To solve the n -queens problem, for example, each of the $n \times n$ board posi-
 131 tions is represented by a neuron whose output is either one or zero depending
 132 on whether a queen is currently placed in that position or not. (Note that
 133 this is a local representation rather than a distributed representation of the
 134 board.) If two board positions are inconsistent, then an inhibiting connection
 135 exists between the corresponding two neurons. For example, all the neurons
 136 in a column will inhibit each other, representing the constraint that two
 137 queens cannot be in the same column. For each row, there is a guard neuron
 138 connected to each of the neurons in that row which gives the neurons in the
 139 row a large excitatory input, enough so that at least one neuron in the row
 140 will turn on. The guard neurons thus enforce the constraint that one queen

141 in each row must be on. As described above, the network is updated on each
142 cycle by randomly picking a row and flipping the state of the neuron in that
143 row whose input is most inconsistent with its current output. A solution is
144 realized when the output of every neuron is consistent with its input.

145 3 Why does the GDS Network Perform So 146 Well?

147 Our analysis of the GDS network was motivated by the following question:
148 “Why does the network perform so much better than traditional backtracking
149 methods on certain tasks”? In particular, we were intrigued by the results on
150 the n -queens problem, since this problem has received considerable attention
151 from previous researchers. For n -queens, Adorf and Johnston found empir-
152 ically that the network requires a linear number of transitions to converge.
153 Since each transition requires linear time, the expected (empirical) time for
154 the network to find a solution is $O(n^2)$. To check this behavior, Johnston
155 and Adorf ran experiments with n as high as 1024, at which point memory
156 limitations became a problem.¹

157 3.1 Nonsystematic Search Hypothesis

158 Initially, we hypothesized that the network’s advantage came from the non-
159 systematic nature of its search, as compared to the systematic organization
160 inherent in depth-first backtracking. There are two potential problems as-
161 sociated with systematic depth-first search. First, the search space may be
162 organized in such a way that poorer choices are explored first at each branch
163 point. For instance, in the n -queens problem, depth-first search tends to find

¹The 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).

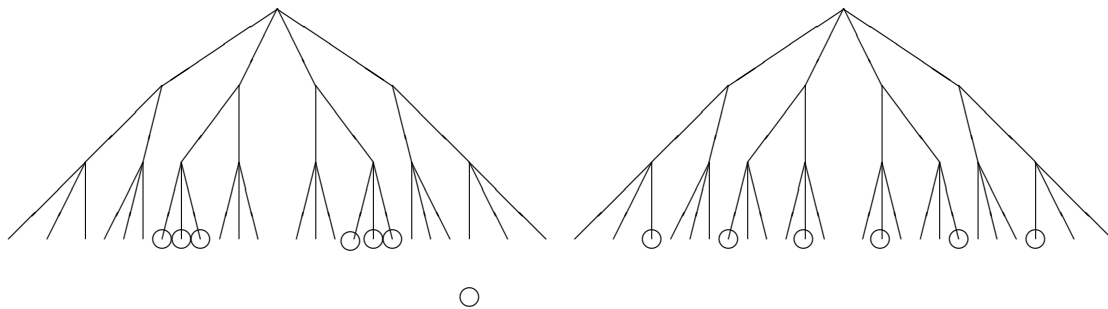


Figure 1: Solutions Clustered vs. Solutions Evenly Distributed

164 a solution more quickly when the first queen is placed in the center of the
 165 first row rather than in the corner; apparently this occurs because there are
 166 more solutions with the queen in the center than with the queen in the corner
 167 Stone and Stone (1987). Nevertheless, most naive algorithms tend to start
 168 in the corner simply because humans find it more natural to program that
 169 way. However, this fact by itself does not explain why nonsystematic search
 170 would work so well for n -queens. A backtracking program that randomly
 171 orders rows (and columns within rows) performs much better than the naive
 172 method, but still performs poorly relative to the GDS network.

173 The second potential problem with depth-first search is more significant
 174 and more subtle. As illustrated by figure 1, a depth-first search can be
 175 a disadvantage when solutions are not evenly distributed throughout the
 176 search space. In the tree at the left of the figure, the solutions are clustered
 177 together. In the tree on the right, the solutions are more evenly distributed.
 178 Thus, the average distance between solutions is greater in the left tree. In a
 179 depth-first search, the average time to find the first solution increases with the
 180 average distance between solutions. Consequently depth-first search performs
 181 relatively poorly in a tree where the solutions are clustered, such as that on
 182 the left Ginsberg and Harvey (1990); Langley (1992). In comparison, a search
 183 strategy which examines the leaves of the tree in random order is unaffected
 184 by solution clustering.

185 We investigated whether this phenomenon explained the relatively poor
186 performance of depth-first search on n -queens by experimenting with a ran-
187 domized search algorithm, called a Las Vegas algorithm Brassard and Bratley
188 (1988). The algorithm begins by selecting a path from the root to a leaf. To
189 select a path, the algorithm starts at the root node and chooses one of its
190 children with equal probability. This process continues recursively until a
191 leaf is encountered. If the leaf is a solution the algorithm terminates, if not,
192 it starts over again at the root and selects a path. The same path may be
193 examined more than once, since no memory is maintained between successive
194 trials.

195 The Las Vegas algorithm does, in fact, perform better than simple depth-
196 first search on n -queens Brassard and Bratley (1988). However, the perfor-
197 mance of the Las Vegas algorithm is still not nearly as good as that of the
198 GDS network, and so we concluded that the systematicity hypothesis alone
199 cannot explain the network's behavior.

200 3.2 Informedness Hypothesis

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

213 **Min-Conflicts heuristic:**

214 *Given:* A set of variables, a set of binary constraints, and an assign-

215 ment specifying a value for each variable. Two variables *conflict* if
216 their values violate a constraint.
217 *Procedure:* Select a variable that is in conflict, and assign it a value
218 that minimizes the number of conflicts. (Break ties randomly.)

219 We have found that the network’s behavior can be approximated by a
220 symbolic system that uses the min-conflicts heuristic for hill climbing. The
221 hill-climbing system starts with an initial assignment generated in a prepro-
222 cessing phase. At each choice point, the heuristic chooses a variable that is
223 currently in conflict and reassigns its value, until a solution is found. The
224 system thus searches the space of possible assignments, favoring assignments
225 with fewer total conflicts. Of course, the hill-climbing system can become
226 “stuck” in a local maximum, in the same way that the network may become
227 “stuck” in a local minimum. In the next section we present empirical evi-
228 dence to support our claim that the min-conflicts approach can account for
229 the network’s effectiveness.

230 There are two aspects of the min-conflicts hill-climbing method that dis-
231 tinguish it from standard CSP algorithms. First, instead of incrementally
232 constructing a consistent partial assignment, the min-conflicts method *re-*
233 *pairs* a complete but inconsistent assignment by reducing inconsistencies.
234 Thus, it uses information about the current assignment to guide its search
235 that is not available to a standard backtracking algorithm. Second, the use
236 of a hill-climbing strategy rather than a backtracking strategy produces a
237 different style of search.

238 3.2.1 Repair-Based Search Strategies

239 (This is a example of a third level section.) Extracting the method from the
240 network enables us to tease apart and experiment with its different compo-
241 nents. In particular, the idea of repairing an inconsistent assignment can be
242 used with a variety of different search strategies in addition to hill climbing.
243 For example, we can backtrack through the space of possible repairs, rather
244 than using a hill-climbing strategy, as follows. Given an initial assignment

```

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

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 omitted]

5 A Theoretical Model

[section omitted]

6 Discussion

[section omitted]

268 7 Acknowledgement

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

274 Appendix A. Probability Distributions for N- 275 Queens

276 [section omitted]

277 References

- 278 Adorf, H. and Johnston, M. (1990). A discrete stochastic neural network
279 algorithm for constraint satisfaction problems. In *Proceedings of the Inter-*
280 *national Joint Conference on Neural Networks*.
- 281 Brassard, G. and Bratley, P. (1988). *Algorithmics - Theory and Practice*.
282 Englewood Cliffs, NJ: Prentice Hall.
- 283 Ginsberg, M. and Harvey, W. (1990). Iterative broadening. In *Proceedings of*
284 *AAAI-91*.
- 285 Hopfield, J. (1982). Neural networks and physical systems with emergent
286 collective computational abilities. In *Proceedings of the National Academy*
287 *of Sciences*, vol. 79, Washington, DC: National Academy Press.
- 288 Johnson, D., Papadimitrou, C. and Yannakakis, M. (1988). How easy is local
289 search? *Journal of Computer and System Sciences*, 37, pp. 79–100.
- 290 Johnston, M. (1987). Automated telescope scheduling. In *Proceedings of the*
291 *Symposium on Coordination of Observational Projects*.

292 Johnston, M. and Adorf, H. (1989). Learning in stochastic neural networks
 293 for constraint satisfaction problems. In *Proceedings of NASA Conference*
 294 *on Space Telerobotics*, vol. 37.

295 Langley, P. (1992). Systematic and nonsystematic search strategies. In *Pro-*
 296 *ceedings of AAAI-92*.

297 Minton, S., Johnston, M., Philips, A. and Laird, P. (1990). Solving large scale
 298 constraint satisfaction and scheduling problems using a heuristic repair
 299 method. In *Proceedings of AAAI-90*.

300 Simmons, R. (1988). A theory of debugging plans and interpretations. In
 301 *Proceedings of AAAI-88*.

302 Stone, H. and Stone, J. (1987). Efficient search techniques - an empirical
 303 study of the n-queens problem. *IBM Journal of Research and Development*,
 304 31, pp. 464–474.

305 Sussman, G. J. (1975). *A Computer Model of Skill Acquisition*. New York:
 306 New American Elsevier.

307 Waldrop, M. (1989). Will the Hubble space telescope compute? *Science*, 243,
 308 pp. 1437–1439.