

1 An empirical comparison of the structure of
2 configuration files of continuous integration and
3 build systems

4 Joseph Ling
jl653@kent.ac.uk



School of Computing
University of Kent
United Kingdom

Word Count: 6,100

5 January 7, 2020

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

1 Introduction

For continuous integration (CI) and build systems a lot of it is configured with code as configuration. The main kind of configuration format used for this is yaml followed by xml and special edge cases. As CI is increasingly becoming more popular and practical ().....For software development teams there are roles for just looking after and managing CI and the deployment infrastructure. Understanding how code as configuration is used for CI is an emerging area of research because of this.

As Travis CI is the most popular research has already be done on patterns found in the structure Gallaba and McIntosh (2018). We will follow a similar methodology in order to get our data to analyse.

Although we won't be looking at puppet () research has also be done on the "code smell" which can be an indication on the quality of the code by analyzing its structure Sharma, Fragkoulis and Spinellis (2016).

Initially this paper started off looking at how to visualize Continuous Integration (CI) and Continuous delivery (CD) systems. However due to their being a lack of research in the area () on specifically on how CI is used and practical implementation. A lot of research has been done supporting the use of CI/CD and it's implementation and benefits in cases for particular languages.

2 Previous Works

By almost any measure, the Hubble Space Telescope scheduling problem 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 prob-

51 lem; it was estimated that it would take over three weeks for the system to
 52 schedule one week of observations. As described in section 4, this problem
 53 was remedied by the development of a successful constraint-based system to
 54 augment the initial system. At the heart of the constraint-based system is
 55 a neural network developed by Adorf and Johnston, the Guarded Discrete
 56 Stochastic (GDS) network,

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

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

76 To illustrate the network architecture and updating scheme, let us con-
 77 sider how the network is used to solve binary constraint satisfaction problems.
 78 A problem consists of n variables, $X_1 \dots X_n$, with domains $D_1 \dots D_n$, and a
 79 set of binary constraints. Each constraint $C_\alpha(X_j, X_k)$ is a subset of $D_j \times D_k$
 80 specifying incompatible values for a pair of variables. The goal is to find
 81 an assignment for each of the variables which satisfies the constraints. (In

82 this paper we only consider the task of finding a single solution, rather than
83 that of finding all solutions.) To solve a CSP using the network, each vari-
84 able is represented by a separate set of neurons, one neuron for each of the
85 variable’s possible values. Each neuron is either “on” or “off”, and in a solu-
86 tion state, every variable will have exactly one of its corresponding neurons
87 “on”, representing the value of that variable. Constraints are represented by
88 inhibitory (i.e., negatively weighted) connections between the neurons. To
89 insure that every variable is assigned a value, there is a guard neuron for
90 each set of neurons representing a variable; if no neuron in the set is on, the
91 guard neuron will provide an excitatory input that is large enough to turn
92 one on. (Because of the way the connection weights are set up, it is unlikely
93 that the guard neuron will turn on more than one neuron.) The network is
94 updated on each cycle by randomly picking a set of neurons that represents
95 a variable, and flipping the state of the neuron in that set whose input is
96 *most inconsistent* with its current output (if any). When all neurons’ states
97 are consistent with their input, a solution is achieved.

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

113 3 Why does the GDS Network Perform So 114 Well?

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

125 3.1 Nonsystematic Search Hypothesis

126 Initially, we hypothesized that the network’s advantage came from the non-
127 systematic nature of its search, as compared to the systematic organization
128 inherent in depth-first backtracking. There are two potential problems as-
129 sociated with systematic depth-first search. First, the search space may be
130 organized in such a way that poorer choices are explored first at each branch
131 point. For instance, in the n -queens problem, depth-first search tends to find
132 a solution more quickly when the first queen is placed in the center of the
133 first row rather than in the corner; apparently this occurs because there are
134 more solutions with the Queen in the center. Nevertheless, most naive algorithms tend to start
135 in the corner simply because humans find it more natural to program that
136 way. However, this fact by itself does not explain why nonsystematic search
137 would work so well for n -queens. A backtracking program that randomly

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).

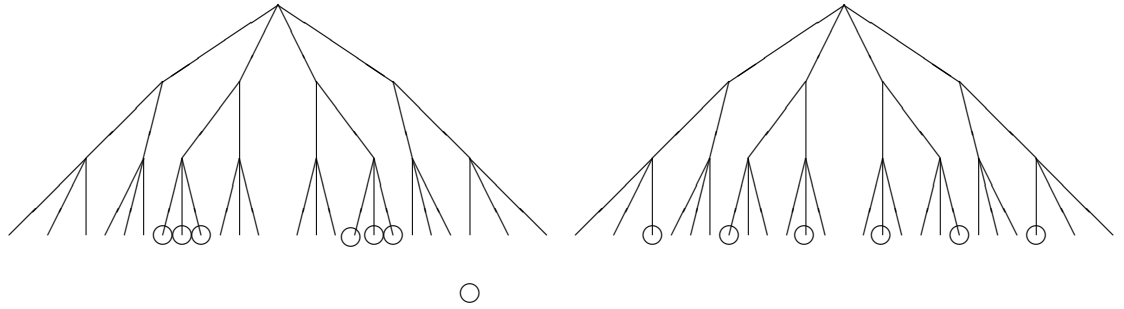


Figure 1: Solutions Clustered vs. Solutions Evenly Distributed

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 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 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

159 trials.

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

165 3.2 Informedness Hypothesis

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

178 **Min-Conflicts heuristic:**

179 *Given:* A set of variables, a set of binary constraints, and an assign-
180 ment specifying a value for each variable. Two variables *conflict* if
181 their values violate a constraint.

182 *Procedure:* Select a variable that is in conflict, and assign it a value
183 that minimizes the number of conflicts. (Break ties randomly.)

184 We have found that the network's behavior can be approximated by a
185 symbolic system that uses the min-conflicts heuristic for hill climbing. The
186 hill-climbing system starts with an initial assignment generated in a prepro-
187 cessing phase. At each choice point, the heuristic chooses a variable that is

188 currently in conflict and reassigns its value, until a solution is found. The
189 system thus searches the space of possible assignments, favoring assignments
190 with fewer total conflicts. Of course, the hill-climbing system can become
191 “stuck” in a local maximum, in the same way that the network may become
192 “stuck” in a local minimum. In the next section we present empirical evi-
193 dence to support our claim that the min-conflicts approach can account for
194 the network’s effectiveness.

195 There are two aspects of the min-conflicts hill-climbing method that dis-
196 tinguish it from standard CSP algorithms. First, instead of incrementally
197 constructing a consistent partial assignment, the min-conflicts method *re-*
198 *pairs* a complete but inconsistent assignment by reducing inconsistencies.
199 Thus, it uses information about the current assignment to guide its search
200 that is not available to a standard backtracking algorithm. Second, the use
201 of a hill-climbing strategy rather than a backtracking strategy produces a
202 different style of search.

203 3.2.1 Repair-Based Search Strategies

204 (This is a example of a third level section.) Extracting the method from the
205 network enables us to tease apart and experiment with its different compo-
206 nents. In particular, the idea of repairing an inconsistent assignment can be
207 used with a variety of different search strategies in addition to hill climbing.
208 For example, we can backtrack through the space of possible repairs, rather
209 than using a hill-climbing strategy, as follows. Given an initial assignment
210 generated in a preprocessing phase, we can employ the min-conflicts heuristic
211 to order the choice of variables and values to consider, as described in figure
212 2. Initially, the variables are all on a list of VARS-LEFT, and as they are
213 repaired, they are pushed onto a list of VARS-DONE. The algorithm attempts
214 to find a sequence of repairs, such that no variable is repaired more than
215 once. If there is no way to repair a variable in VARS-LEFT without violat-
216 ing a previously repaired variable (a variable in VARS-DONE), the algorithm
217 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 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

218 Notice that this algorithm is simply a standard backtracking algorithm
219 augmented with the min-conflicts heuristic to order its choice of which vari-
220 able and value to attend to. This illustrates an important point. The back-
221 tracking repair algorithm incrementally extends a consistent partial assign-
222 ment (i.e., VARS-DONE), as does a constructive backtracking program, but
223 in addition, uses information from the initial assignment (i.e., VARS-LEFT)
224 to bias its search. Thus, it is a type of *informed backtracking*. We still char-
225 acterize it as repair-based method since its search is guided by a complete,
226 inconsistent assignment.

227 4 Experimental Results

228 [section ommitted]

229 5 A Theoretical Model

230 [section ommitted]

231 6 Discussion

232 [section ommitted]

233 7 Acknowledgement

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

239 Appendix A. Probability Distributions for N- 240 Queens

241 [section omitted]

242 References

- 243 Gallaba, K. and McIntosh, S. (2018). Use and Misuse of Continuous Inte-
244 gration Features: An Empirical Study of Projects that (mis)use Travis CI.
245 *IEEE Transactions on Software Engineering*, pp. 1–1.
- 246 Sharma, T., Fragkoulis, M. and Spinellis, D. (2016). Does Your Configuration
247 Code Smell? In *2016 IEEE/ACM 13th Working Conference on Mining*
248 *Software Repositories (MSR)*, pp. 189–200, iSSN: null.