

1 Usage and structure of continuous integra-  
2 tion as configuration?

3 Joseph Ling  
j1653@kent.ac.uk



School of Computing  
University of Kent  
United Kingdom

Word Count: 6,100

4 February 25, 2020

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

<https://arxiv.org/ftp/arxiv/papers/1703/1703.07019.pdf>

Continuous integration (CI) is becoming more popular over the last few years. This can be seen by how major version control hosting services Github, Bitbucket and Gitlab have all started to or have been improving their CI product. In terms of research, configuration as code Rahman, Mahdavi-Hezaveh and Williams (2019) and continuous integration Copeland (2010) with Shahin, Ali Babar and Zhu (2017) demonstrating breadth of the research.

Continuous integration is a process of automatically running compiling, running tests and checking that the product works. This can be combined with Continuous Delivery where the product is deployed or released after it has gone through CI.

This can get complicated quickly therefore configuration as code (or infrastructure as code) is used to configure it. The main kind of configuration format used for this is yaml (reference to what it is??) followed by xml and java based scripting formats.

In terms of looking at usage we are going to do a similar look at the data as did Michael Hilton, Marinov and Dig (2016). The important aspect will be looking at how usage has changed over the last 5 years along with looking more closely at which repositories are more likely to use CI/CD. For this we are going to focus on the following research questions:

- usage of CI vs non usage
- multiple CI used
- per language CI usage
- stars, subscribers and commits for likelihood of using CI

This should give us a better understanding of the sample of repositories from Github. From there we look at the structure of the configuration files to understand how certain aspects of it are used.

- 48 • configuratizon errors when loading the config (just yaml parsing errors  
49 atm)
- 50 • how are comments used in the configuration?
- 51 • how scripts with the configuration files? (need to elloborate more on  
52 this one)

53 A key aspect is that these questions do not look too deeply into the  
54 individual implementation of each CI system. This is because there are  
55 already some good papers looking Gallaba and McIntosh (2018) at this but  
56 in order to be able to compare the different configuration types it is important  
57 to compare similar attributes (there is also a time factor in here as well).

## 58 2 Previous Works

59 Configuration as code or infrastructure as code has been an increasing area  
60 of research over the last few years. There seems to be slightly more research  
61 in infrastructure as code Rahman, Mahdavi-Hezaveh and Williams (2019).  
62 The has been a focus on Puppet and Chef, for example in Sharma, Fragkoulis  
63 and Spinellis (2016) looks at code quality by the measure of "code smell" of  
64 Puppet code. This tackles the problem by defining by best practices and  
65 analyzing the code against that. In the case of Cito et al. (2017) it uses  
66 the docker linter in order to be able to analyse the files. For the continous  
67 inteграtion systems we pick we will look into the tooling around that to aid  
68 the analysis.

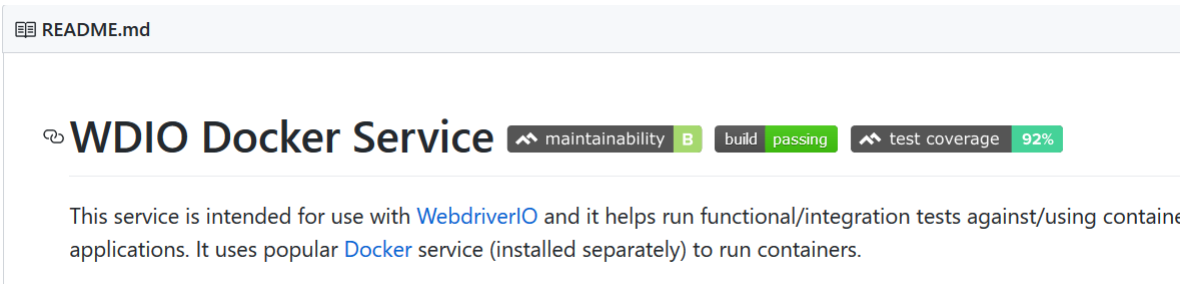
69 Michael Hilton, Marinov and Dig (2016)

## 70 3 Methodology

71 In order to get repositories with CI/CD configuration from Github we have a  
72 number of approaches. The first is too use the search for particular files but  
73 this is limited to only 1000 results. The alternative is to search for repositories

74 and we bypass the 1000 result limit to an extent by getting results for every  
75 'star' count (stars are used to like or upvote a repository). Although this will  
76 be giving us a lot of results it will still only be a sample of the population but  
77 will give us a wider range of results. As there is rate limiting multiple github  
78 api keys can be used to speed up the scraping of data (gitorrent could also  
79 be used to speed up the process I think).

80 After we have got a repository we need to get the CI/CD files from it.  
81 This is fairly easy as the CI/CD systems normally require a strict naming  
82 convention and location within the repository. However as most of them are  
83 yaml based you can have ".yaml" and ".yml" and users can use all sorts of  
84 mixtures of upper and lower case. We try to account for this but won't get  
85 every scenario. This combined with the fact that we are only looking for  
86 top configuration files based on github (2017) along with github actions and  
87 azure pipelines. Is why we also check repositories for their ReadMe.md file  
88 to check if it has a build tag.



89

where did this im-90  
age come from??  
reference it man 91

91 In doing so it should give a wider net when sampling and help to under-  
92 stand when a CI system is either not using configuration as code or using a  
93 different CI system.

94 There are dangers in scraping data off github in terms of assumptions to  
95 do with the population as found in Kalliamvakou et al. (2014). In Github  
96 you can fork a repository which copies in order to remove these we check for  
97 fork flag on the repository. This causes are dataset to go from NUMBER to  
98 OTHER\_NUMBER.

99 Additionally the assumption that all repositories are of programming

100 projects with code in them is wrong. A number of repositories can be used  
101 for storage, experimental, academic and other things. However they to all  
102 some extent can use CI/CD for their work as a number of books were found  
103 when looking through the dataset could use CI/CD.

105 Tooling for the configuration files, I looked into Travis, Github Actions  
106 and Jenkins to work out whether or not it could aid in the research or not.  
107 As a key part of understanding the first relies on knowing whether or not  
108 it is valid. In terms for travis there is currently two parsers to validate the  
109 configuration. One which is depracted since 2017 ([https://github.com/travis-](https://github.com/travis-ci/travis-yaml/)  
110 [ci/travis-yaml/](https://github.com/travis-ci/travis-yaml/)) the other which is currently in development ([https://github.com/travis-](https://github.com/travis-ci/travis-ci)  
111 [ci/travis-yml](https://github.com/travis-ci/travis-ci)). Both didn't provided the necessary results with the most re-  
112 cent one not being able to handle default fields. For Github Actions as it's  
113 still a new tooling for it hasn't been developed outside of the Github editor  
114 web page ([https://github.community/t5/GitHub-Actions/YAML-validator-](https://github.community/t5/GitHub-Actions/YAML-validator-for-Github-Actions-possible-expansion-of/td-p/29557)  
115 [for-Github-Actions-possible-expansion-of/td-p/29557](https://github.community/t5/GitHub-Actions/YAML-validator-for-Github-Actions-possible-expansion-of/td-p/29557)). For Jenkins which is  
116 older solution allows validation through http/ssh request to the Jenkins server  
117 (Gitlab follows this style as well) Jenkins (2020) Gitlab (2020). This could  
118 work well although would require setting up a server for each configuration  
119 type and might not validate if varaibles from the config aren't defined on the  
120 server. As well as it would be best to be able to validate them all or none of  
121 them in terms of being able to compare results easily.

Analyse of readme can be used to try and classify but I don't think that is necessary. I think the key factor is that any concluding remarks anywhere take this into ac-

Additionally the paper suggests looking a for re-cent commits but I don't think that is necessary just yet... but would be a good indication later on size/dating the projects when comparing CI/CD systems.

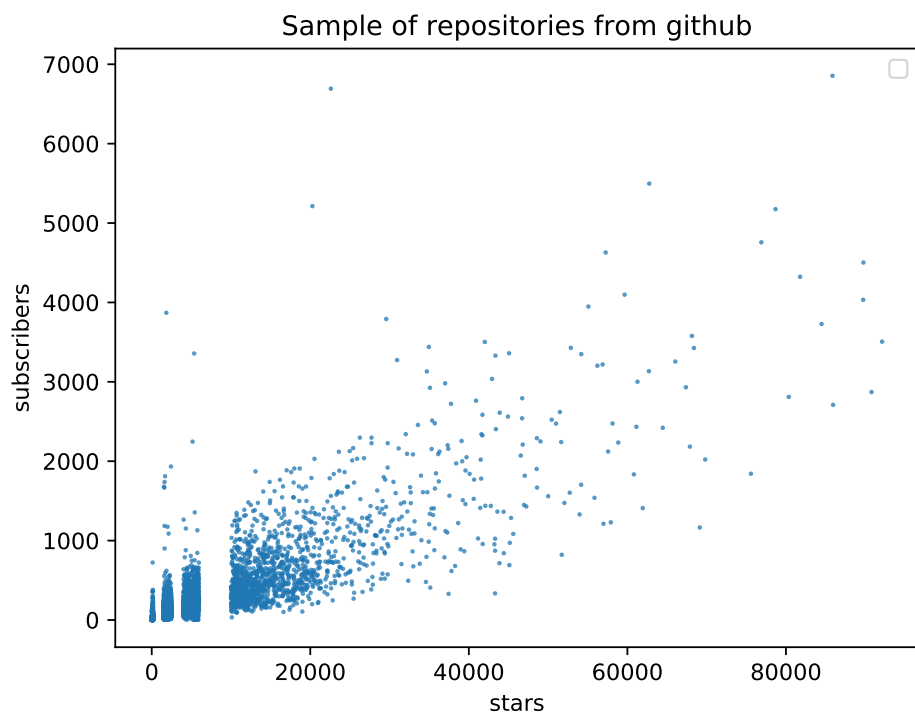
## 122 4 Usage of CI

### 123 4.1 configuration errors when loading the config (just 124 yaml parsing errors atm)

125 This leads us to get the following data:

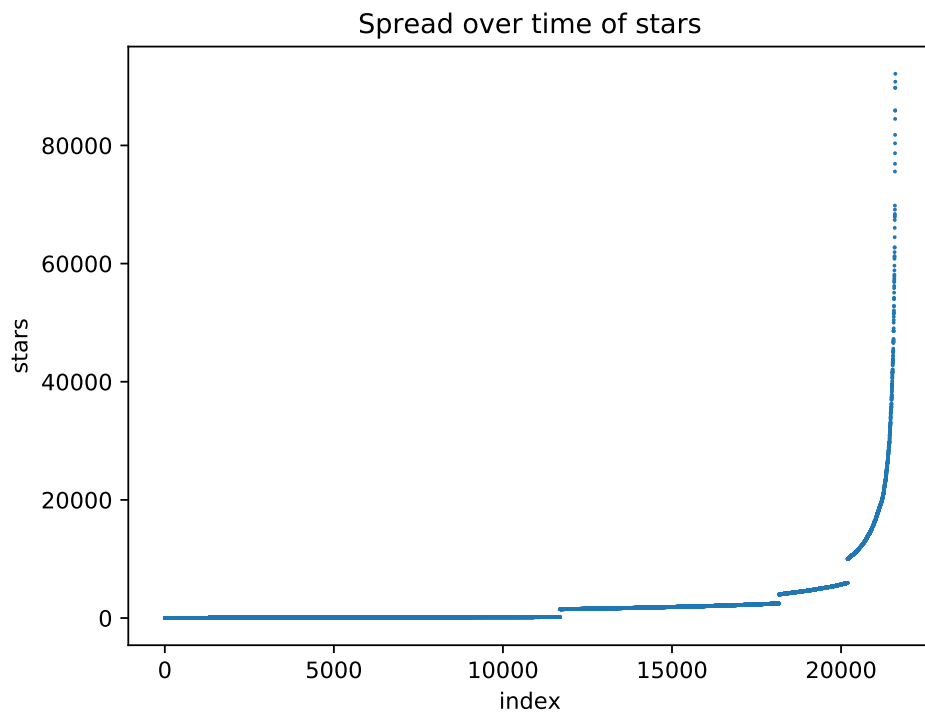
126 A repository on github is like a folder so can contain any number of  
127 configuration files in it. Therefore we can get any number of configuration  
128 files in that folder. This is taken into account with the second pair of columns  
129 for the first row. It demonstrates that their a large number of repositories

130 with multiple kinds of configuration (todo make sure that github actions  
131 multiple file thing isn't calculated here).  
132     The next row is for when we couldn't pick up the configuration used for  
133 CI/CD and check the ReadMe.md file for build status tag.  
134     The final row is shows the repositories that either don't have any config-  
135 uration or no configuration that could be found.  
136     However that doesn't give us too much insight into the dataset. Here is a  
137 graph showing the subscribers plotted against the number of stars. The key  
138 here to understand is not potentially any correlation but to see the spread  
139 of data that the table is showing.





141       The second graph helps give a understanding to the give a depth of the  
142 data for where the graph is just blue. This is because on Github you get  
143 more repositories with smaller star counts than large ones.



CI/CD	count	configs per repo	duplicates	duplicate percent
config file	8327	39%	1221	15%
found in ReadMe	582	3%		
none found	11469	53%		

145 - what is the spread of ci/cd systems for public github repositories? this  
146 will take into account operating system, programming language, star count,  
147 subscriber count - note: something along the lines of multiple configuration  
148 files - what naming convention do they use for the files? (in order to under-  
149 stand common practices)

150 this will follow on from the previous graphs looking at spread of CI/CD  
151 configs found in the whole sample

152 then look at the difference that large repositories more than 100 commits  
153 with more than 2 contributors

154 then look at recent commits

155 perhaps a small look at naming conventions used???

## 156 5 Config file results

157 5.1 configuration errors when loading the config (just  
158 yaml parsing errors atm)

159 5.2 How are comments used in configuration?

160 5.3 How are stages used in configuration?

161 and looking into branches

162 5.4 How are script tags used?

163 - how scripts with the configuration files? (need to elaborate more on this  
164 one)

165 By almost any measure, the Hubble Space Telescope scheduling problem  
166 Between ten thousand and thirty thousand astronomical observations per

	config	percentage
travis	7051	74%
github	1544	16%
circleci	759	8%
jenkinsPipeline	113	1%
drone	54	1%
buildkite	20	0%
teamcity	4	0%
azure	1	0%
semaphore	1	0%

yaml_encoding_error	composer error	constructor error	parse error	scanner error
circleci	1	0	0	1
drone	20	0	0	0
github	0	1	0	2
travis	4	0	6	16

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 7, 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,

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

181 addition to the space telescope scheduling problem. For example, the network  
182 performs significantly better on the  $n$ -queens problem than methods that  
183 were previously developed. The  $n$ -queens problem requires placing  $n$  queens  
184 on an  $n \times n$  chessboard so that no two queens share a row, column or diagonal.  
185 The network has been used to solve problems of up to 1024 queens, whereas  
186 most heuristic backtracking methods encounter difficulties with problems  
187 one-tenth

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

198     To illustrate the network architecture and updating scheme, let us con-  
199 sider how the network is used to solve binary constraint satisfaction problems.  
200 A problem consists of  $n$  variables,  $X_1 \dots X_n$ , with domains  $D_1 \dots D_n$ , and a  
201 set of binary constraints. Each constraint  $C_\alpha(X_j, X_k)$  is a subset of  $D_j \times D_k$   
202 specifying incompatible values for a pair of variables. The goal is to find  
203 an assignment for each of the variables which satisfies the constraints. (In  
204 this paper we only consider the task of finding a single solution, rather than  
205 that of finding all solutions.) To solve a CSP using the network, each vari-  
206 able is represented by a separate set of neurons, one neuron for each of the  
207 variable's possible values. Each neuron is either "on" or "off", and in a solu-  
208 tion state, every variable will have exactly one of its corresponding neurons  
209 "on", representing the value of that variable. Constraints are represented by  
210 inhibitory (i.e., negatively weighted) connections between the neurons. To  
211 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.

## 6 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

241 from previous researchers. For  $n$ -queens, Adorf and Johnston found empir-  
242 ically that the network requires a linear number of transitions to converge.  
243 Since each transition requires linear time, the expected (empirical) time for  
244 the network to find a solution is  $O(n^2)$ . To check this behavior, Johnston  
245 and Adorf ran experiments with  $n$  as high as 1024, at which point memory  
246 limitations became a problem.<sup>1</sup>

## 247 6.1 Nonsystematic Search Hypothesis

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

262 The second potential problem with depth-first search is more significant  
263 and more subtle. As illustrated by figure 1, a depth-first search can be  
264 a disadvantage when solutions are not evenly distributed throughout the  
265 search space. In the tree at the left of the figure, the solutions are clustered  
266 together. In the tree on the right, the solutions are more evenly distributed.

---

<sup>1</sup>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).

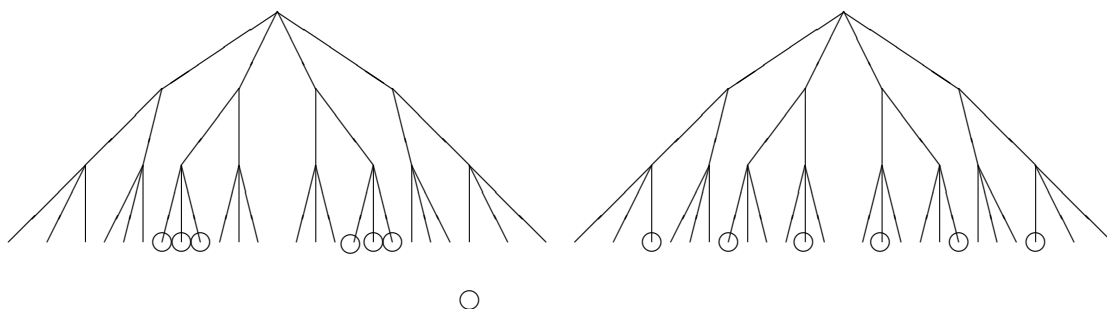


Figure 1: Solutions Clustered vs. Solutions Evenly Distributed

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 solutions are clustered. 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. 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.



## 287 6.2 Informedness Hypothesis

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

### 300 **Min-Conflicts heuristic:**

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

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

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

```

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

317 There are two aspects of the min-conflicts hill-climbing method that dis-  
 318 tinguish it from standard CSP algorithms. First, instead of incrementally  
 319 constructing a consistent partial assignment, the min-conflicts method *re-*  
 320 *pairs* a complete but inconsistent assignment by reducing inconsistencies.  
 321 Thus, it uses information about the current assignment to guide its search  
 322 that is not available to a standard backtracking algorithm. Second, the use  
 323 of a hill-climbing strategy rather than a backtracking strategy produces a  
 324 different style of search.

### 325 6.2.1 Repair-Based Search Strategies

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

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

## 349 7 Experimental Results

350 [section ommitted]

## 351 8 A Theoretical Model

352 [section ommitted]

## 353 9 Discussion

354 [section ommitted]

## 355 10 Acknowledgement

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

## 361 Appendix A. Probability Distributions for N- 362 Queens

363 [section ommitted]

## 364 References

- 365 Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S. and Gall,  
366 H. C. (2017). An Empirical Analysis of the Docker Container Ecosystem  
367 on GitHub. In *2017 IEEE/ACM 14th International Conference on Mining*  
368 *Software Repositories (MSR)*, pp. 323–333, iSSN: null.
- 369 Copeland, P. (2010). Google’s Innovation Factory: Testing, Culture, and  
370 Infrastructure. In *Proceedings of the 2010 Third International Conference*

371     *on Software Testing, Verification and Validation*, Washington, DC, USA:  
372     IEEE Computer Society, ICST '10, pp. 11–14.

373     Gallaba, K. and McIntosh, S. (2018). Use and Misuse of Continuous Inte-  
374     gration Features: An Empirical Study of Projects that (mis)use Travis CI.  
375     *IEEE Transactions on Software Engineering*, pp. 1–1.

376     github (2017). <https://github.blog/2017-11-07-github-welcomes-all-ci-tools/>.  
377     In github.com, ed., *github welcomes all ci tools*.

378     Gitlab (2020). <https://docs.gitlab.com/ee/api/lint.html>. In *Gitlab docs*.

379     Jenkins (2020). <https://jenkins.io/doc/book/pipeline/development/>. In  
380     *Jenkins documentation*.

381     Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M. and  
382     Damian, D. (2014). The promises and perils of mining GitHub. Hyderabad,  
383     India: Association for Computing Machinery, MSR 2014, pp. 92–101.

384     Michael Hilton, K. H., Timothy Tunnell, Marinov, D. and Dig, D. (2016).  
385     Usage, costs, and benefits of continuous integration in open-source projects  
386     | Proceedings of the 31st IEEE/ACM International Conference on Auto-  
387     mated Software Engineering.

388     Rahman, A., Mahdavi-Hezaveh, R. and Williams, L. (2019). A systematic  
389     mapping study of infrastructure as code research. *Information and Soft-*  
390     *ware Technology*, 108, pp. 65–77.

391     Shahin, M., Ali Babar, M. and Zhu, L. (2017). Continuous Integration, De-  
392     livery and Deployment: A Systematic Review on Approaches, Tools, Chal-  
393     lenges and Practices. *IEEE Access*, 5, pp. 3909–3943.

394     Sharma, T., Fragkoulis, M. and Spinellis, D. (2016). Does Your Configuration  
395     Code Smell? In *2016 IEEE/ACM 13th Working Conference on Mining*  
396     *Software Repositories (MSR)*, pp. 189–200, iSSN: null.