

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 February 6, 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

need more introduction into CI like the travis paper probably?

For continuous integration (CI) and build systems/continuous delivery a lot of it is configured with code as configuration. The main kind of configuration format used for this is yaml (reference to what it is??) followed by xml and java based scripting formats. As CI has become increasingly 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.

cite needed here to backup

The first stage in order to answer this question is to get a dataset of configuration files from different ci/cd systems. In doing so answering the questions of: - what is the spread of ci/cd systems for public github repositories? this will take into account operating system, programming language, star count, subscriber count - note: something along the lines of multiple configuration files - what naming convention do they use for the files? (in order to understand common practices)

After getting all that data then it will be need to analysed to how it is used. The key questions for this that will be focused on will be: - how are comments used in the configuration? - how are stages (named sections of the process) named for configuration files? - branch names - configuration errors when loading the config (just yaml parsing errors atm) - how scripts with the configuration files? (need to elloborate more on this one)

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

need to carefully plan out how to gage the size of repo, as I don't want to have too download repos

2 Previous Works

Config as code is not necessarily infrastructure as code is slightly differently. So the comparison isn't fair necessarily or accurate.

Configuration as code or infrastructure as code has been an increasing area of research over the last few years. There seems to be slightly more research in infrastructure as code Rahman, Mahdavi-Hezaveh and Williams (2019). There has been a focus on Puppet and Chef, for example in Sharma, Fragkoulis and Spinellis (2016) looks at code quality by the measure of "code smell" of Puppet code. This tackles the problem by defining by best practices and analyzing the code against that. In the case of Cito et al. (2017) it uses the docker linter in order to be able to analyse the files.

15, 18, 23, 30

docker, puppet, travis analysis

3 Methodology

In order to get repositories with CI/CD configuration from Github we have a number of approaches. The first is to use the search for particular files but this is limited to only 1000 results. The alternative is to search for repositories and we bypass the 1000 result limit to an extent by getting results for every 'star' count (stars are used to like or upvote a repository). Although this will be giving us a lot of results it will still only be a sample of the population but will give us a wider range of results. As there is rate limiting multiple github api keys can be used to speed up the scraping of data (ghTORRENT could also be used to speed up the process I think).

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

WDIO Docker Service

maintainability B

build passing

test coverage 92%

This service is intended for use with [WebdriverIO](#) and it helps run functional/integration tests against/using container applications. It uses popular [Docker](#) service (installed separately) to run containers.

81

where did this image come from??
reference it man

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

In doing so it should give a wider net when sampling and help to understand when a CI system is either not using configuration as code or using a different CI system.

Results and the spread of the actual data with the watches vs stars maybe the star count over time to demonstrate the floors in the data gathering process

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

graph of data again

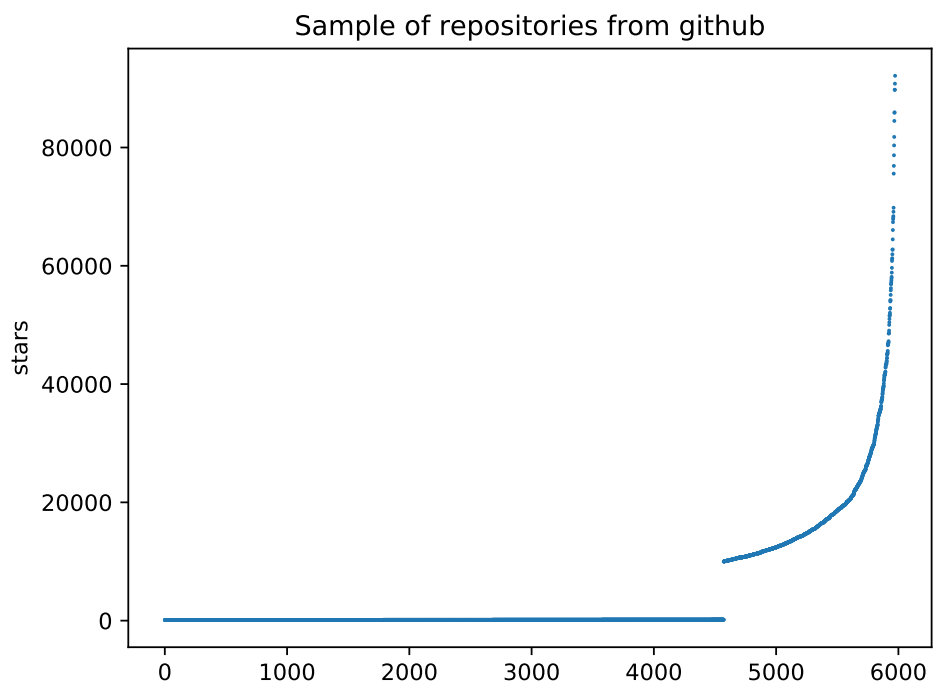
Additionally the assumption that all repositories are of programming projects with code in them is wrong. A number of repositories can be used for storage, experimental, academic and other things. However they to all some extent can use CI/CD for their work as a number of books were found when looking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

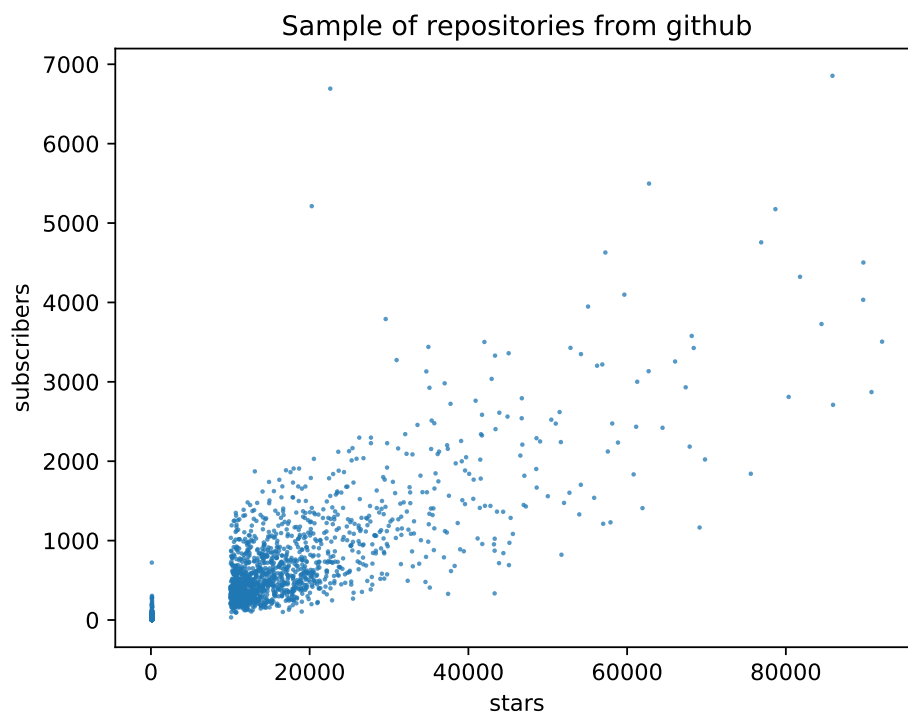
3.1 CI/CD spread in the sample

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

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 account.

Additionally the paper suggests looking a for recent 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.





109 this will follow on from the previous graphs looking at spread of CI/CD
110 configs found in the whole sample
111 then look at the difference that large repositories more than 100 commits
112 with more than 2 contributors
113 then look at recent commits
114 perhaps a small look at naming conventions used???

115 4 Config file results

116 4.1 configuration errors when loading the config (just
117 yaml parsing errors atm)

118 4.2 How are comments used in configuration?

119 4.3 How are stages used in configuration?

120 and looking into branches

121 4.4 How are script tags used?

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

124 By almost any measure, the Hubble Space Telescope scheduling problem
125 Between ten thousand and thirty thousand astronomical observations per
126 year must be scheduled, subject to a great variety of constraints including
127 power restrictions, observation priorities, time-dependent orbital character-
128 istics, movement of astronomical bodies, stray light sources, etc. Because the
129 telescope is an extremely valuable resource with a limited lifetime, efficient
130 scheduling is a critical concern. An initial scheduling system, developed us-
131 ing traditional programming methods, highlighted the difficulty of the prob-
132 lem; it was estimated that it would take over three weeks for the system to
133 schedule one week of observations. As described in section 6, this problem
134 was remedied by the development of a successful constraint-based system to
135 augment the initial system. At the heart of the constraint-based system is

136 a neural network developed by Adorf and Johnston, the Guarded Discrete
137 Stochastic (GDS) network,

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

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

157 To illustrate the network architecture and updating scheme, let us con-
158 sider how the network is used to solve binary constraint satisfaction problems.
159 A problem consists of n variables, $X_1 \dots X_n$, with domains $D_1 \dots D_n$, and a
160 set of binary constraints. Each constraint $C_\alpha(X_j, X_k)$ is a subset of $D_j \times D_k$
161 specifying incompatible values for a pair of variables. The goal is to find
162 an assignment for each of the variables which satisfies the constraints. (In
163 this paper we only consider the task of finding a single solution, rather than
164 that of finding all solutions.) To solve a CSP using the network, each vari-
165 able is represented by a separate set of neurons, one neuron for each of the
166 variable's possible values. Each neuron is either "on" or "off", and in a solu-

tion 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.

194 5 Why does the GDS Network Perform So 195 Well?

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

206 5.1 Nonsystematic Search Hypothesis

207 Initially, we hypothesized that the network’s advantage came from the non-
208 systematic nature of its search, as compared to the systematic organization
209 inherent in depth-first backtracking. There are two potential problems as-
210 sociated with systematic depth-first search. First, the search space may be
211 organized in such a way that poorer choices are explored first at each branch
212 point. For instance, in the n -queens problem, depth-first search tends to find
213 a solution more quickly when the first queen is placed in the center of the
214 first row rather than in the corner; apparently this occurs because there are
215 more solutions with the Queen in the center. Nevertheless, most naive algorithms tend to start
216 in the corner simply because humans find it more natural to program that
217 way. However, this fact by itself does not explain why nonsystematic search
218 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).

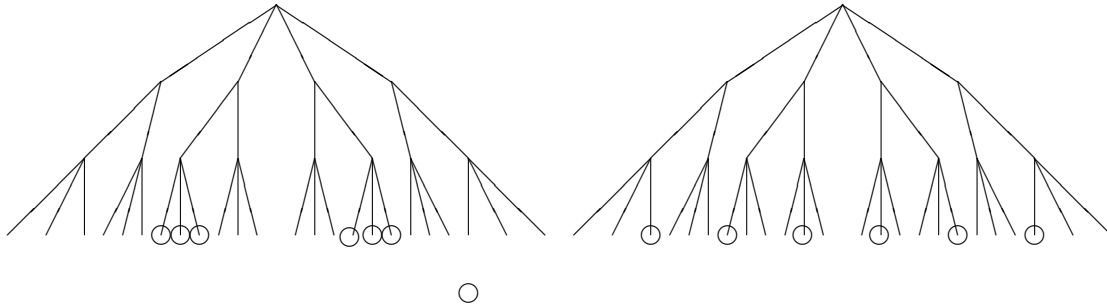


Figure 1: Solutions Clustered vs. Solutions Evenly Distributed

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 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

240 trials.

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

246 5.2 Informedness Hypothesis

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

259 **Min-Conflicts heuristic:**

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

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

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

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

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

5.2.1 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 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.

```

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

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

308 6 Experimental Results

309 [section ommitted]

310 7 A Theoretical Model

311 [section ommitted]

312 8 Discussion

313 [section ommitted]

314 9 Acknowledgement

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

320 Appendix A. Probability Distributions for N- 321 Queens

322 [section omitted]

323 References

- 324 Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S. and Gall,
325 H. C. (2017). An Empirical Analysis of the Docker Container Ecosystem
326 on GitHub. In *2017 IEEE/ACM 14th International Conference on Mining*
327 *Software Repositories (MSR)*, pp. 323–333, iSSN: null.
- 328 Gallaba, K. and McIntosh, S. (2018). Use and Misuse of Continuous Inte-
329 gration Features: An Empirical Study of Projects that (mis)use Travis CI.
330 *IEEE Transactions on Software Engineering*, pp. 1–1.
- 331 github (2017). <https://github.blog/2017-11-07-github-welcomes-all-ci-tools/>.
332 In github.com, ed., *github welcomes all ci tools*.
- 333 Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M. and
334 Damian, D. (2014). The promises and perils of mining GitHub. Hyderabad,
335 India: Association for Computing Machinery, MSR 2014, pp. 92–101.
- 336 Rahman, A., Mahdavi-Hezaveh, R. and Williams, L. (2019). A systematic
337 mapping study of infrastructure as code research. *Information and Soft-*
338 *ware Technology*, 108, pp. 65–77.
- 339 Sharma, T., Fragkoulis, M. and Spinellis, D. (2016). Does Your Configuration
340 Code Smell? In *2016 IEEE/ACM 13th Working Conference on Mining*
341 *Software Repositories (MSR)*, pp. 189–200, iSSN: null.