# Usage and structure of continuous integration as configuration?

Joseph Ling

jl653@kent.ac.uk

School of Computing
University of Kent
United Kingdom

Word Count: 6,100

March 10, 2020

## Abstract

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

# 1 Introduction

Continous integeration (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 integeration Copeland (2010) with Shahin, Ali Babar and Zhu (2017) demonstrating breadth of the research.

Continous integeration is a process of automatically running compiling, running tests and checking that the product works. This is can be combined with Continous 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 do a similar look at the data as did Michael Hilton, Marinov and Dig (2016). The importanat 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:

- What percentange of open-source projects use CI?
- multiple CI used
- what is the breakdown of usage of different services?
- Do certain types of projects use CI more than others?

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.

- configuratizon errors when loading the config (just yaml parsing errors atm)

- how are comments used in the configuration?

- Are external scripts used within the configuration?

A key aspect is that these questions do not look too deeply into the individual implementation of each CI system. 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).

# 2 Previous Works

## 2.1 Continous integeration

Continous integeration is the frequent submission of work normally tied into a feedback loop. For example using version control daily committing changes. That then a server builds and tests the changes informing you of status of those cahnges. The generally agree upon detailed definition is Fowler (2010).

## 2.2 Usage of continous integeration

The actual usage of continous integeration as configuration was looked at by Michael Hilton, Marinov and Dig (2016). In this they use three source of information github repositories, travis builds and a survery. In order to be do a more systematic study of CI usage than Vasilescu et al. (2015). In analysing that data they found that "The trends that we discovered point to an expected growth of CI. In the future, CI will have an even greater influence than it has today.". As we are looking at the same question we will use four of the same research questions out of the fourteen. In order to see what difference four years has made to the growth of usage of CI.

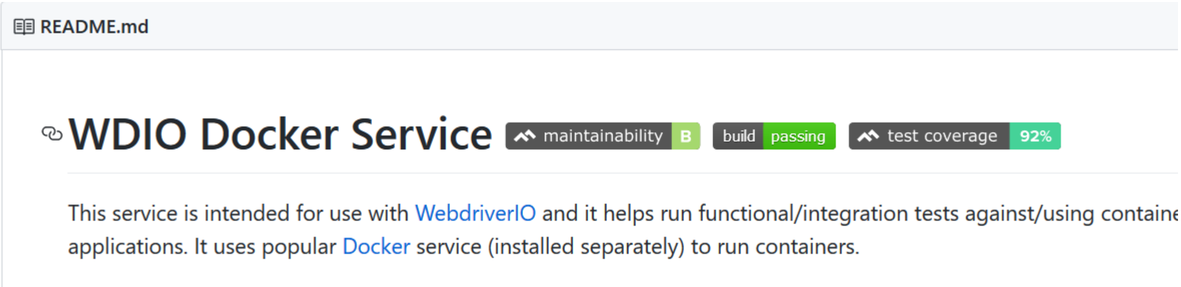
### 2.3 Config as code

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). The 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. For the continous integeration systems we pick we will look into the tooling around that to aid the analysis.

## 3 Methodology

In order to get repositories with CI/CD configuration from Github we have a number of approaches. The first is too 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 their 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 ”.yml” and ”.yaml” 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

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

Figure 1: Example of CI tag for Github ReadMe

where did this image come from?? reference it man

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.

There are dangers in scraping data off github in terms of assumptions to do with the population as found in Kalliamvakou et al. (2014). Our dataset does not contain any forked repositories. But due to time constraints number of commits and frequency of recent commits has not been looked at. This would be an interesting area of further research in order to improve the quality of the sample but also to look at how that affects the frequency of CI usage.

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.

Tooling for the configuration files, I looked into Travis, Github Actions and Jenkins to work out whether or not it could aid in the research or not. As a key part of understanding the first relies on knowing whether or not it is valid. In terms for travis there is currently two parsers to validate the configuration. One which is depracted since 2017 travis (2017) the other which is currently in development travis (2020). Both didn't provided the necessary results with the most recent one not being able to handle default fields. For

Github Actions as it's still a new tooling for it hasn't been developed outside of the Github editor web page (https://github.community/t5/GitHub-Actions/YAML-validator-for-Github-Actions-possible-expansion-of/td-p/29557). For Jenkins which is older solution allows validation through http/ssh request to the Jenkins server (Gitlab follows this style as well) Jenkins (2020) Gitlab (2020). This could work well although would require setting up a server for each configuration type and might not validate if varaibles from the config aren't defined on the server. As well as it would be best to be able to validate them all or none of them in terms of being able to compare results easily.

# 4 Usage of CI

## 4.1 What percentange of open-source projects use CI?

Based a search for configuration as configuration files for the following CI systems: Travis, Gitlab, Azure, App Veyor, Drone, Jenkins, Github, Circleci, Semaphore, Teamcity and buildkite. Wrecker got bought by Oracle and from doing a search on Github for what I think based on the docs (docs: Wrecker and Oracle (2018) and search: GitHub (2020)) for their config file naming convention. I was only able to find 20 results so did not include in the scraping script to speed up the process of searching for the other configuration file formats.

| CI/CD | **count** | **repos with config** | **no. multiple** | **multiple percent** |
|---|---|---|---|---|
| config file(s) | 12128 | 38.51% | 1675 | 13.81% |
| found in ReadMe | 873 | 2.77% | | |
| none found | 18493 | 58.72% | | |

Table 1: Percentage of CI used for projects

Our sample of repositories is 31,494 in comparison to Michael Hilton, Marinov and Dig (2016) which had a sample of 34,544. The percentage of CI projects they had was 40.27%. As if you combined the ”config file(s)” and ”found in ReadMe”. However in order to work out if a project might be using CI but the config file wasn’t picked a search string is used. Therefore it is not as accurate as finding a config file as their could be false postives.

However that doesn’t give us too much insight into the dataset. Here is a graph showing the subscribers plotted against the number of stars. The key here to understand is not potentially any correlation but to see the spread of data that the table is showing.

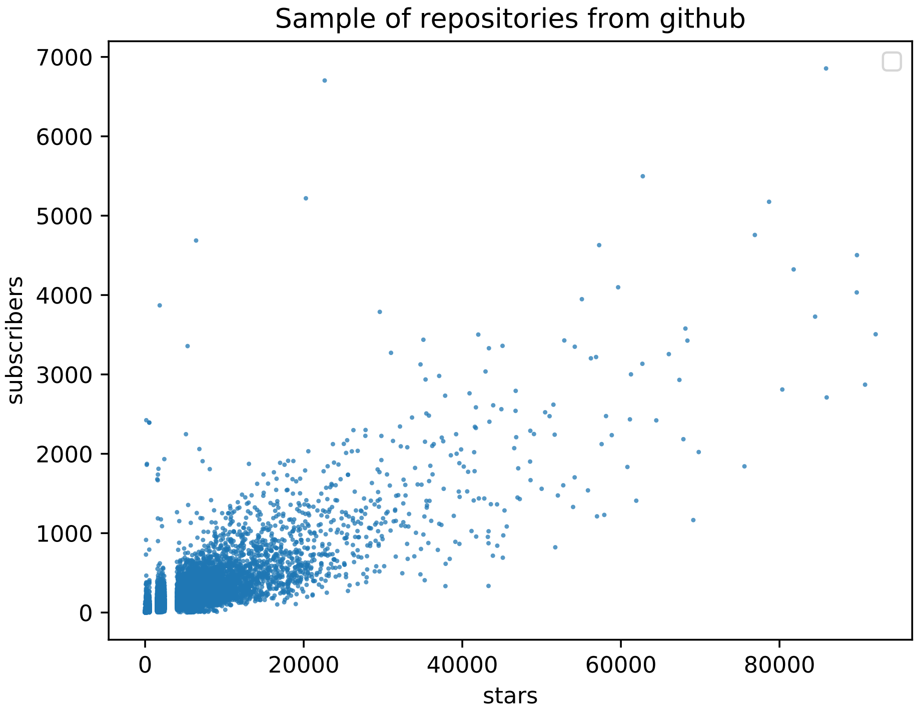


Figure 2: Scatter graph of Github stars against subscribers

Figure 2 helps give a understanding to the give a depth of the data for where the graph is just blue. This is because on Github you get more repositories with smaller star counts than large ones.

Figure 3 provides insight into the density of the data for between 0 to 25000.

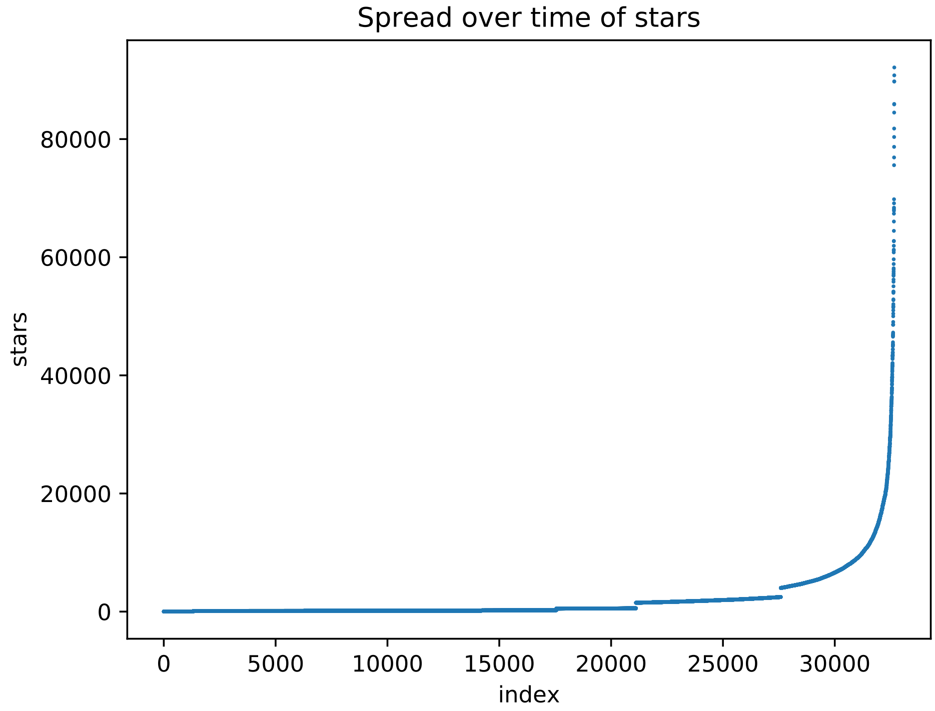


Figure 3: Stars graph

## 4.2 What CI systems are projects using?

Like all other research travis is the most popular CI system in use. However over the last 4 years since the github (2017) Circleci has lost out on it's rough quarter that it owned. In particular the rise of github actions seems to have taken second place even though it is still very young in comparison (DATES). However this might not be down to the Circleci loosing out on their existing share. But potentially as the rise in CI usage goes up on github. Projects are more likely to pick in the built in solutions to github.

Table 2: Configuration types spread

| | config | percentage |
|---|---|---|
| travis | 10607 | 74% |
| github | 2301 | 16% |
| circleci | 1109 | 8% |
| jenkinsPipeline | 161 | 1% |
| drone | 84 | 1% |
| buildkite | 32 | 0% |
| teamcity | 4 | 0% |
| semaphore | 2 | 0% |
| azure | 1 | 0% |

### 4.3 Do certain types of projects use CI more than others?

Below shows all the CI projects sorted then grouped together per 540 projects. Then in this case we choose to categories via star count for each project.

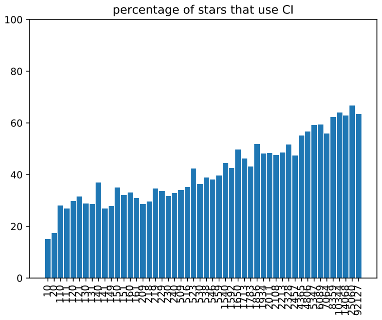


Figure 4: 2020 dataset

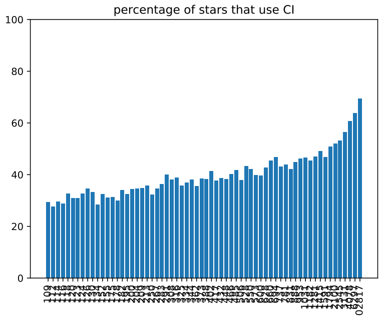


Figure 5: 2016 dataset

Figure 6: In Figure 4 is the results from this research and in Figure 5 is the results from Michael Hilton, Marinov and Dig (2016).

Here we are comparing whether or not in the last 4 years the number of stars increases the CI being used. Their seems to a steeper gradient in the more recent datasets. However as 4 starts at zero stars and 5 starts at 100 stars their is signifacant dip at the start of the first graph.

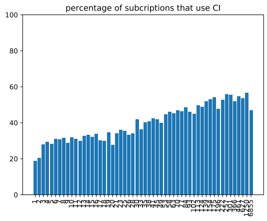


Figure 7: Subs graph

Figure 7 uses the same method as Figure TODO SORT except is does it based the number of subscribers. Subscribers are used on github to keep update on the changes on the project. This ranges from core team memembers working on the project to people that want to be notified about a new release. In looking at this metric the hypothesis was that it would have a sharper rise in percentage of projects using CI per subscriber. However that was not the case overall the gradient is not as strong. There is no comparisson to Michael Hilton, Marinov and Dig (2016) because their final corpus does not contain subscriber count for each project.

# 5 Config file results

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

**Composer error** In the example it has two steps that are using an yaml anchor. This allows for the yaml below it to be referenced somewhere else. However if you define the anchor twice it causes a composer error. As you have two references for the samething so it won't know which one to use.

```
definitions:
steps:
- step: &build-test
name: Build and test
script:
- mvn package
- step: &build-test
name: deploy
script:
- ./deploy.sh target/my-app.jar
```

**Scanner error** The first step of loading the yaml is to scan it to create the tokens. However invalid characters such as "\t" are invalid.

```
definitions: \t
```

As can be seen in the table their our configuration files with yaml errors meaning that the CI for that project will not load. Yet it seems that a very small percentage of projects that have them. For example the two

**Parse error** In this example it has scanned the file and created tokens for the syntax. Now it parses the syntax and works out if each token is valid given it's current context. In this case a closing ] without an opening [ is invalid.

```
definitions: ]
```

Table 3: yaml configuration errors

| **config** | composer error | constructor error | parse error | scanner error | no. config |
|---|---|---|---|---|---|
| **circleci** | 1 | 0 | 0 | 1 | 1109 |
| **drone** | 31 | 0 | 0 | 0 | 84 |
| **github** | 0 | 1 | 0 | 3 | 2301 |
| **travis** | 6 | 0 | 10 | 21 | 10607 |
| **buildkite** | 0 | 0 | 0 | 0 | 32 |
| **semaphore** | 0 | 0 | 0 | 0 | 2 |
| **azure** | 0 | 0 | 0 | 0 | 1 |

highest configuration types with errors are drone (36.90%) followed by travis (0.348%).

In the case for drone all the errors are for the same type of error. Potentailly this could be because of how anchors are a lot more common in drone.

For travis it is the most common form of CI found therefore it is more likely to contain more errors. Yet with such a small amount it seems like yaml errors aren't a major problem in CI. Although as they are required to be fixed in order for the CI to run the chances of it working are higher and a more detailed study would need to be done.... ah

## 5.2 How are comments used in configuration?

The assumption was the as continuous integeration setups can be complicated and have edge cases. Therefore comments would be used to describe and handle that complexity.

An example configuration file below for Github actions using the default template slightly altered. Shows two examples of comment usage, the first being including useful information about why a particular version of the programming language was chosen. The second is that the tests have been disabled by commenting them out.

In order to pick up on all these different types of comments. All the CI files were parsed and then regular expressions were used to pick on up key factors such as ”note:”. Along with multiple single line comments which made up a block/multi-line comment.
For example in to the left there is an example Github Action yaml file. If were it would be parsed we would get: one multi line comment, 15 lines of code, 1 single line comment, a total of 5 comments and 20 lines in the file. Therefore their is a their is a raito of 4:1 for code in this config file.

```
name: Python package
on: [push]
jobs:
build:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v2
- name: Set up Python
uses: actions/setup-python@v1
# note: only works with python 3
with:
python-version: 3.8
- name: Install dependencies
run: |
python -m pip install --upgrade pip
pip install -r requirements.txt
#      - name: Test with pytest
#        run: |
#          pip install pytest
#          pytest ./src
```

Initally before we look at the comments it is important to understand how the rest of the file is made up. In the graph below (Figure 8) it shows how each configuration type is made up by mean of each part of the file. For all the yaml based configurations lines of code and number of lines in total are very close. Then for the number of commments being very very small on average.

In the case for Jenkins pipelines and teamcity there is a much higher usage of having code with commments.

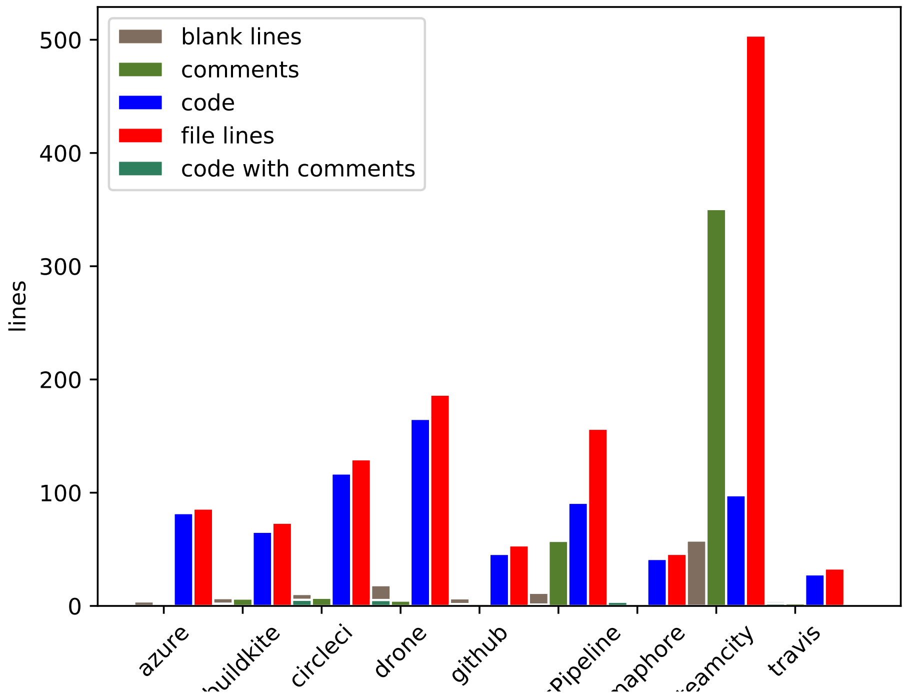


Figure 8: Mean of line counts

Raitos:

- code: comments
- code: line total
- code: blank lines
- single line comment: multiline comment
- single line comment: code with comment

In Figure 9 a regular expression was used to label the comments. There were key different types of comment that we wanted to find. The first being the commented out code which we did by searching for version numbers in commments. The second being useful information about the structure of the CI file such todo, note, importanat comments (e.g. //todo). In order to increase the search for this we included searching for urls and seperation comments (e.g. //===).

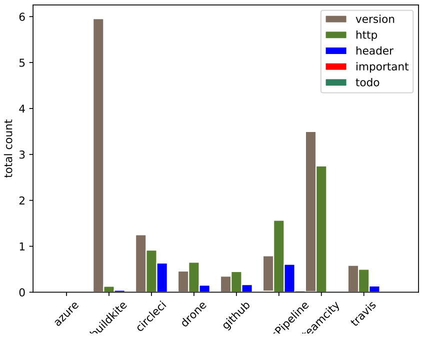


Figure 9: Comment types

From labelling the comments in Figure 9 we can see that having commments with versions in and urls is most common. This could indicate comments from templates or how they are commented. Although yet again the amount of labels found on average is still very low.

Overall we have found that comments are not used a lot. In the cases that they are used it's more likely to be from a configuration template or commenting out configuration.

### 5.3 Are external scripts used within the configuration?

An external script is a bash or powershell script typically depending on the operating system. It can be used to build, deploy or do any step that CI takes. The key difference between it and the CI configutation is that it be executed on a users machine. Therefore you do get some setups where you have scripts defined for building and deploying the code that the users and CI both use. Most CI systems allow for "script" tags to be used which could be descibed as an internal script. Therefore external scripts are defined outside the CI configuration in the directory.

The methodology we used to handle this was too look at how many bash or powershell scripts where used in CI. Using the code the parsed the yaml files for comments we were able to check do a using a regular expression for either of those files.

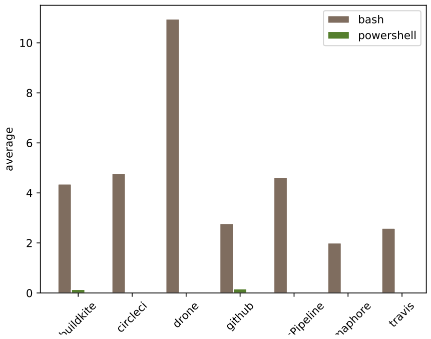


Figure 10: Comment types

| | bash | powershell |
|---|---|---|
| **buildkite** | 61 | 2 |
| **circleci** | 1497 | 8 |
| **drone** | 230 | 0 |
| **github** | 1097 | 65 |
| **jenkinsPipeline** | 171 | 0 |
| **semaphore** | 2 | 0 |
| **travis** | 5937 | 3 |

Figure 11: sum of scripts used

In Figure 10 we have the average number of times a script is used for a configuration file that already has a script being used.

As some of the necessary actions are being done in the scripts and not in the CI file. Potentailly there could be less lines of code in the configuration for files that use scripts. However in Figure 12 we can see that the data is all over the place.

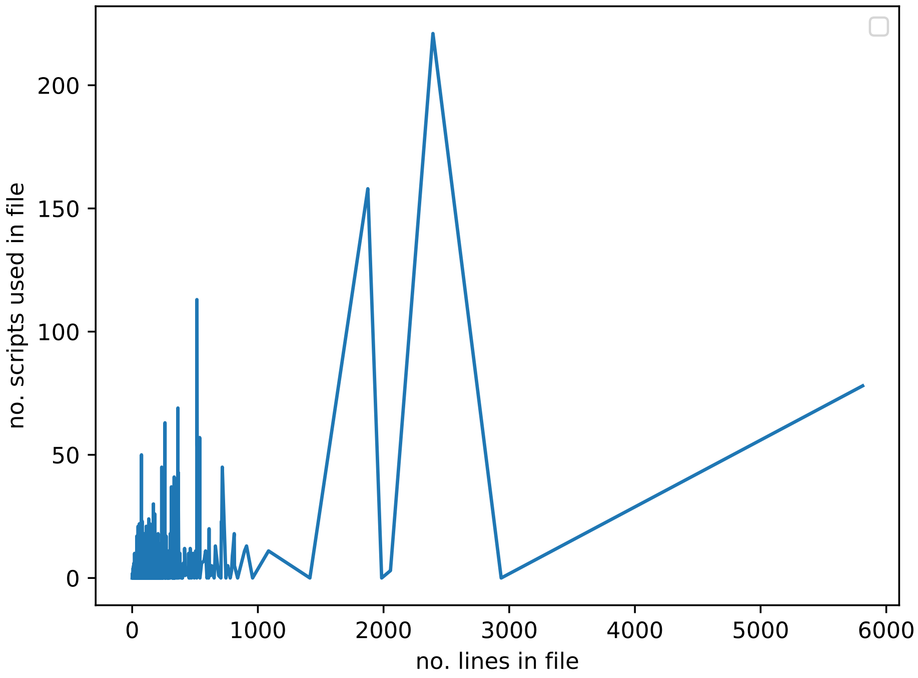


Figure 12: Comment types

To conclude we can see that scripts are used but that much. And their no correlation between lines of code and usage of external scripts.

# 6 Threats to validatity

The major and most obivious threat is the sample gathered from scraping the data from Github. This has already been touched on in the 3 section but now we are going to look at it in more detail.

Firstly if we assume that the scraping works perfectly then it's only at maximum a 1000 open source projects per star. That is excluding closed source projects which would range from personal projects to companies. As well as it is only data from Github not from gitlab, bitbucket or other version control hosting services. This leads to bais in the data for example if gitlab was also scraped then we would get a lot more gitlab ci files. However in order to get best spread of data Github has the best api and most servies do not tie you down to use only their service. As well although we could get a 1000 projects per star we were still able to get around 30,000 projects and a wide spread across Github. The key aspect being that because it was a sample we focused on getting a good spread of data.

Secondly the scraping script is not perfect in how it finds configuration files. As it only looks in the top level directory for the file name pattern described in their docs or unique folder. Therefore if the systems allowed many different names or different names in past it wouldn't have picked it CI system. Additionally we only decided to scrape for certain CI files. Yet we chose a good scope based on previous research into the top CI files. As well the scraping script has been tested worked on to try and minimise any bugs. In the case that we did not pick up a CI file we ran a regexp against the ReadMe file to get a better understanding of the error bounds.

Thirdly identifying which projects are programming projects or would have a need for CI. Based on the research Kalliamvakou et al. (2014) it is important to filter out repositories that aren't part of the question being asked. Therefore we could have looked to try and filter out github static sites and other none software based projects. However if assume a certain type of project won't be using CI then we would be introducing bais when trying to answer how CI is used. For further research better labelling of what kind of projects are which would potentailly benefical though.

# 7 Summary

asdfasdf should be another 400 words

## 7.1 Discussion and further research

In the process of writing this paper we kept on considering more research questions. As there is a lot of meta data that you can get for a single project, in addition to what was used for this paper.

Further research into usage that we would like to do is look into how the size of the project affects the chance that it uses CI. Then looking at the usage of scripts within CI configuration, for example using a script tag to run a shell script. As while doing the research we found some projects use scripts a lot while others just used the CI config. This would lead to questions around which CI system has a higher amount of scripts used. But also looking at how much they enable them to be used and what is the size of those scripts. The data for the programming language and version(s) is in the config. Therefore it would be possible to work out how much usage each version is getting of a particular programming language.

Further research into structure could look into the naming of each part of the build process that is used. This would be interesting as it would provided insight into what terms are commonly used. As well an idea into how people plan or don't plan out their configuration files. Additionally CI systems can be designed to run on every commit to version control or only commits to certain branches. Therefore by looking at the branching regexp that are being used an better understanding of how branches are actually used in software development where CI is also used could be found out.

In addition working on pruning our dataset using methods outlined in Kalliamvakou et al. (2014).

# 8 Acknowledgement

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

# Appendix A. Probability Distributions for N-Queens

[section ommitted]

# References

Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S. and Gall, H. C. (2017). An Empirical Analysis of the Docker Container Ecosystem on GitHub. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 323–333, iSSN: null.

Copeland, P. (2010). Google's Innovation Factory: Testing, Culture, and Infrastructure. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation*, Washington, DC, USA: IEEE Computer Society, ICST '10, pp. 11–14.

Fowler, M. (2010). Continous integration. In *https://www.martinfowler.com/articles/continuousIntegration.html*.

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

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

GitHub (2020). github filename search for wrecker.yml files. In *github filename search for wrecker.yml files*.

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

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

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

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

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

Shahin, M., Ali Babar, M. and Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, 5, pp. 3909–3943.

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

travis (2017). travis yaml (old repository). In *https://github.com/travis-ci/travis-yaml/*.

travis (2020). travis yaml new implementation. In *https://github.com/travis-ci/travis-yml/*.

Vasilescu, B., Yu, Y., Wang, H., Devanbu, P. and Filkov, V. (2015). Quality and productivity outcomes relating to continuous integration in GitHub. Bergamo, Italy: Association for Computing Machinery, ESEC/FSE 2015, pp. 805–816.

Wrecker and Oracle (2018). Wrecker ci development blog. In *Wrecker CI development blog*.