On the Prediction of Continuous Integration Build Failures Using Search-Based Software Engineering

Islem Saidani
École de Technologie Supérieure
Montreal, QC, Canada
islem.saidani.1@ens.etsmtl.ca

Ali Ouni
École de Technologie Supérieure
Montreal, QC, Canada
ali.ouni@etsmtl.ca

Moataz Chouchen
École de Technologie Supérieure
Montreal, QC, Canada
moataz.chouchen.1@ens.etsmtl.ca

Mohamed Wiem Mkaouer
Rochester Institute of Technology
Rochester, NY, USA
mwmvse@rit.edu

ABSTRACT

Continuous Integration (CI) aims at supporting developers in integrating code changes quickly through automated building. However, in such context, the build process is typically time and resource-consuming. As a response, the use of machine learning (ML) techniques has been proposed to cut the expenses of CI build time by predicting its outcome. Nevertheless, the existing ML-based solutions are challenged by problems related mainly to the imbalanced distribution of successful and failed builds. To deal with this issue, we introduce a novel approach based on Multi-Objective Genetic Programming (MOGP) to build a prediction model. Our approach aims at finding the best prediction rules based on two conflicting objective functions to deal with both minority and majority classes. We evaluated our approach on a benchmark of 15,383 builds. The results reveal that our technique outperforms state-of-the-art approaches by providing a better balance between both failed and passed builds.

CCS CONCEPTS

• **Software and its engineering → Integrated and visual development environments**; **Software configuration management and version control systems**.

KEYWORDS

Continuous integration, build prediction, multi-objective optimization, search-based software engineering, machine learning

ACM Reference Format:
Islem Saidani, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. 2020. On the Prediction of Continuous Integration Build Failures Using Search-Based Software Engineering. In *Genetic and Evolutionary Computation Conference Companion (GECCO '20 Companion), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3377929.3390050

GECCO '20 Companion, July 8–12, 2020, Cancún, Mexico

ACM ISBN 978-1-4503-7127-8/20/07.
https://doi.org/10.1145/3377929.3390050

1 INTRODUCTION

Continuous integration (CI) [3] is a software engineering practice that provides support to developers by automating the testing and building process. However, the adoption of CI is not without barriers. Build failure represents a blocker that prevents developers from proceedings further with development, as it requires an immediate action to resolve it. In addition, the build resolution may take hours or even days to complete, which severely affects both, the speed of software development and the productivity of developers. In the light of these problems, many research efforts [4, 9, 14–16] were conducted to predict, using Machine Learning (ML) techniques, when a software state is most likely to trigger a failure when built. However, none of existing works accommodated for the imbalanced distribution of the successful and failed classes when building their prediction models. Therefore, such models are generally not in full compliance with the real world situation where the failed builds are much less to occur than passed ones [16]. However, in CI context, a good accuracy on the failed builds prediction is more important than the passed builds accuracy. Also, increasing the accuracy of the builds failure class (known as probability of detection) can result in maximizing also the number of incorrectly classified failed builds (*i.e.*, false alarms) which makes these two objectives in conflict [8].

To deal with the above mentioned challenges, we propose a novel a Multi-Objective Genetic Programming (MOGP) approach to predict CI build failure. The idea is based on the adaption of Non-dominated Sorting Genetic Algorithm (NSGA-II) [2] to generate a prediction rule expressed as a combination of CI-related metrics and their appropriate threshold values; and should cover as much as possible the build results using the history of builds.

2 PROPOSED APPROACH

Our adaptation to the NSGA-II algorithm is to adopt it following a tree-like representation instead of fixed length linear string formed from a limited alphabet of symbols. In our problem formulation, a candidate solution, *i.e.*, a prediction rule, is represented as an IF-THEN clause which describes the conditions under which a build is said to be succeeded or failed [5, 10–12]. The condition corresponds to a logical expression that combines some metrics and their threshold values using logical operators (OR, AND). A solution is encoded as a tree where each terminal belongs to the set of predefined CI related metrics (extracted from literature) and their

corresponding thresholds are generated randomly. Each internal-node belongs to the connective set C = {AND, OR}. All the generated solutions are evaluated using two objectives to (1) maximize the true positive rate (TPR), and (2) minimize the false positive rate (FPR). Change operators are applied, at every iteration, to generate new solutions. After repeating this process until reaching a stop criteria, the best solution *i.e.* prediction rule is returned by the algorithm.

3 EXPERIMENTS

3.1 Settings

We conduct an empirical study on a benchmark of 15,383 build instances of two projects, Vagrant[1] and Ruby[2], that use Travis CI system[3]. This benchmark is extracted from TravisTorrent dataset [1]. In this paper, we compare our MOGP approach against:

- Random Search (RS) in order to evaluate the need for an intelligent method such as NSGA-II.
- Genetic Algorithm (GA) to determine if considering separate conflicting objectives to be optimized (multi-objective) is an appropriate formulation compared to aggregating them in a single objective.
- Decision Tree (DT) a widely used ML technique in CI build failure prediction [6, 13, 14]

In order to make our results comparable with the state-of-the-art approaches, we compute the well-known evaluation metrics Area Under the ROC Curve (*AUC*) as well as *balance* [7] to assess the performance of models that were initially trained using imbalanced training data considering online validation. Similar to prior work [14], we ranked for the two selected projects, the builds according to its start time and broke the whole set of a given project into ten folds. Then, we used the latter five folds as testing sets: At each iteration i ($1 \leq i \leq 5$), the test set fold j ($6 \leq j \leq 10$), the former j-1 folds were selected as training set to train the model.

3.2 Results

Table 1 reports the obtained experimental results. We observe that GA and RS achieved on average comparable scores of 65% and 66% respectively in terms of AUC, while GA was slightly better in terms of balance with a score of 58% compared to 54% achieved by RS. We clearly see that NSGA-II outperformed mono-objective algorithms by an increase of 9% and 16% in terms of AUC and balance respectively. This provides evidence that the use of multi-objective formulation for the prediction problem is more suited as it can provide a better compromise between TPR and FPR.

Compared to DT, we can see that our NSGA-II technique shows better predictive performance by achieving, on average, AUC of 75% compared to 63%. Also, in terms of balance, our approach outperformed DT with a score of 74% on average while we recorded a score of 55% for DT. Therefore, we can conjecture that NSGA-II performs better in comparison to the state-of-the-art ML technique without features scaling or relying on any re-sampling technique. This result indicates that our approach is able to strike a better balance between both failed and passed builds accuracies by the means of the multi-objective formulation.

Table 1: Online Validation results.

	AUC				balance			
	DT	*RS*	*NSGA2*	*GA*	*DT*	*RS*	*NSGA2*	*GA*
mitchellh/vagrant	0.65	0.68	**0.77**	0.71	0.56	0.60	**0.76**	0.64
ruby/ruby	0.60	0.61	**0.72**	0.62	0.55	0.48	**0.72**	0.51
Average	0.63	0.65	**0.75**	0.66	0.55	0.54	**0.74**	0.58

4 CONCLUSIONS AND FUTURE WORK

We introduced a new search-based approach for CI build failure prediction. In our genetic programming (GP) adaptation, prediction rules are represented as a combination of metrics and threshold values that should correctly predict as much as possible the failed builds extracted from a base of real world examples. Considering online validation, the obtained results provides evidence that our approach outperforms Decision Tree (DT), for which we applied re-sampling, as well as Random Search and mono-objective Genetic Algorithm, based on a benchmark of 15,383 CI builds. As a future work, we plan also to extend our approach by considering a large base of CI projects as well as comparing our approach to other ML techniques.

REFERENCES

[1] M. Beller, G. Gousios, and A. Zaidman. 2017. TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration. In *EEE/ACM International Conference on Mining Software Repositories*. 447–450.

[2] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2, 182–197.

[3] Martin Fowler. 2006. Continuous Integration. https://www.martinfowler.com/articles/continuousIntegration.html,. (2006). Accessed: 2020-01-01.

[4] Foyzul Hassan and Xiaoyin Wang. 2017. Change-aware build prediction model for stall avoidance in continuous integration. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 157–162.

[5] M. Kessentini and A. Ouni. 2017. Detecting android smells using multi-objective genetic programming. In *IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 122–132.

[6] Y. Luo, Y. Zhao, W. Ma, and L. Chen. 2017. What are the Factors Impacting Build Breakage?. In *Web Information Systems and Applications Conference*. 139–142.

[7] Ruchika Malhotra. 2015. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing* 27 (2015), 504–518.

[8] Ruchika Malhotra and Megha Khanna. 2017. An exploratory study for software change prediction in object-oriented systems using hybridized techniques. *Automated Software Engineering* 24, 3 (2017), 673–717.

[9] Ansong Ni and Ming Li. 2017. Cost-effective build outcome prediction using cascaded classifiers. In *Int. Conference on Mining Software Repositories*. 455–458.

[10] Ali Ouni, Raula Gaikovina Kula, Marouane Kessentini, and Katsuro Inoue. 2015. Web service antipatterns detection using genetic programming. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 1351–1358.

[11] Ali Ouni, Marouane Kessentini, Katsuro Inoue, and Mel O Cinnéide. 2017. Search-based web service antipatterns detection. *IEEE Transactions on Services Computing* 10, 4 (2017), 603–617.

[12] Ali Ouni, Marouane Kessentini, Houari Sahraoui, and Mounir Boukadoum. 2013. Maintainability defects detection and correction: a multi-objective approach. *Automated Software Engineering* 20, 1 (2013), 47–79.

[13] Mark Santolucito, Jialu Zhang, Ennan Zhai, and Ruzica Piskac. 2018. Statically Verifying Continuous Integration Configurations. *Technical Report* (2018).

[14] Jing Xia and Yanhui Li. 2017. Could we predict the result of a continuous integration build? An empirical study. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 311–315.

[15] Jing Xia, Yanhui Li, and Chuanqi Wang. 2017. An Empirical Study on the Cross-Project Predictability of Continuous Integration Outcomes. In *2017 14th Web Information Systems and Applications Conference (WISA)*. IEEE, 234–239.

[16] Zheng Xie and Ming Li. 2018. Cutting the Software Building Efforts in Continuous Integration by Semi-Supervised Online AUC Optimization.. In *IJCAI*. 2875–2881.

[1] https://github.com/hashicorp/vagrant
[2] https://github.com/ruby/ruby
[3] https://travis-ci.org/