Multi-objective Black-Box Test Case Prioritization Based on Wordnet Distances

Imara van Dinten(✉), Andy Zaidman, and Annibale Panichella(✉)

Delft University of Technology, Delft, The Netherlands
{I.vanDinten,A.E.Zaidman,A.Panichella}@tudelft.nl

Abstract. Test case prioritization techniques have emerged as effective strategies to optimize this process and mitigate the regression testing costs. Commonly, black-box heuristics guide optimal test ordering, leveraging information retrieval (e.g., cosine distance) to measure the test case distance and sort them accordingly. However, a challenge arises when dealing with tests of varying granularity levels, as they may employ distinct vocabularies (e.g., name identifiers). In this paper, we propose to measure the distance between test cases based on the shortest path between their identifiers within the WordNet lexical database. This additional heuristic is combined with the traditional cosine distance to prioritize test cases in a multi-objective fashion. Our preliminary study conducted with two different Java projects shows that test cases prioritized with WordNet achieve larger fault detection capability ($APFD_C$) compared to the traditional cosine distance used in the literature.

Keywords: Empirical software engineering · Search-based software testing · Test case prioritization · Wordnet · Natural language processing

1 Introduction

Regression testing is the process of retesting a software project to evaluate whether changes in the production code have any unintended effects on the unchanged portions [12]. The simplistic approach to regression testing involves executing the entire test suite within a CI/CD pipeline. However, this strategy may not be feasible for systems that require extensive resources or have large test suites. Therefore, researchers have proposed different techniques to reduce the cost of regression testing [12], including removing redundant tests, selecting a subset of tests for execution, and sorting the test cases to detect regression faults earlier through test case prioritization (TCP).

Black-box TCP techniques have emerged as effective strategies to optimize this process and mitigate the regression testing costs [8]. The main advantage of black-box heuristics is that they do not require access to the source code and, thus, can be applied to any software project. Black-box heuristics commonly guide optimal test ordering by leveraging diversity/distance metrics. The idea is

P. Arcaini et al. (Eds.): SSBSE 2023, LNCS 14415, pp. 101–107, 2024.
https://doi.org/10.1007/978-3-031-48796-5_7

that diverse test cases cover different parts/behaviors of the system under test and, thus, would be more effective in detecting faults [12].

To this aim, researchers have reduced the TCP problem into a traditional information retrieval (IR) task [8,10], where test cases are treated as textual *queries* and used to retrieve the more diverse (less related) test cases in the test suite. However, these methods statically inspect test case keywords and compare them using well-known similarity functions, such as cosine similarity, hamming distance, etc. Static keyword comparison might only partially capture the semantic distance between test cases if they employ different vocabularies. While unit-level tests that target the same classes may share common identifiers or keywords, system-level tests invoke components and call specific APIs.

To address these limitations, we proposed to leverage WordNet [7], a large lexical database of English words interlinked with each other through semantic and lexical relations. While two words can have different meanings, they may share a common ancestor in the WordNet hierarchy. For example, the words "chapter" and "paragraph" are not identical, but they are semantically related as they are both part of a "document". Our intuition is that WordNet can provide additional information about how two test cases can be related to one another.

Our paper introduces `TestScheduler`, a novel black-box TCP method that combines the traditional cosine distance with the WordNet distances. Our approach relies on multi-objective evolutionary algorithms (and NSGA-II [5] in particular) to prioritize the test cases based on (1) their cosine distance, (2) their WordNet distance, and (3) past test execution cost.

We evaluated `TestScheduler` on two open-source Java projects, namely `Javaparser` and `Apache Commons Lang`. The former is a parser for Java source code, while the latter is part of the Defects4J dataset [6]. We analyze the fault detection capability produced by `TestScheduler`. And compare its performance against a baseline that relies on the traditional IR distance and past execution cost. Our results indicate that TCP by `TestScheduler` can detect more faults than the baseline while incurring a lower execution cost. This confirms our conjecture that WordNet can complement the traditional IR-based methods.

2 Test Case Prioritization Based on WordNet

WordNet [7] is a well-established lexical database of English words grouped in sets of synonyms, called *synsets*. Synsets are connected via synonym relationships, i.e., distinct synsets that share the same meaning. In addition to synonyms, WordNet also includes additional relationships between synsets, such as hypernyms (superordinate terms), hyponyms (subordinate terms), meronyms (part-whole relationships), and holonyms (whole-part relationships). We proposed a novel approach, called `TestScheduler`, that prioritizes test cases based on their semantic distance in WordNet. We elaborate on its main steps below.

Pre-processing. Before using WordNet, the test cases undergo a pre-/processing phase aiming to extract words to query on WordNet and exclude programming language-specific keywords that do not contribute to the test semantics. `TestScheduler` pre-processes the tests by applying various IR steps: (i) *tokenization*, *removing stop words*, and (ii) *stemming*. First, *tokenization* extracts words in the tests and removes non-relevant characters. After which, the compound names are split into tokens [1]. Further, it applies the *stop-word list* function to remove words that do not contribute to the semantic content of the tests [1]. Our stop-list includes the standard list for the English language (e.g., prepositions and articles) [1], plus a list of words that are specific to the programming languages (i.e., keywords like `class` in `Java`). The stop-word function also removes words that contain less than three characters [4]. Finally, *stemming* algorithms reduce the words to their root form using the Porter stemming algorithm [1].

WordNet Distance. Given two test cases t_i and t_j, we measure their semantic distance as the average pairwise distance between their composing words/synsets (after pre-processing) in the WordNet taxonomy/database:

$$\mathrm{WD}(t_a, t_b) = \frac{1}{m \times n} \sum_{i=1}^{m} \sum_{j=1}^{n} d(w_i, w_j) \tag{1}$$

where w_i is the i-th word in t_a; w_j is the j-th word in t_b; m and n are the number of words in t_a and t_b, respectively; $d(w_i, w_j)$ denotes the distance between the words/synsets w_i and w_j in the WordNet taxonomy/graph.

Multiple distances have been defined for the WordNet database, such as the simple *path distance*, the *Leacock Chodorow* similarity, *Resnik*, and the *Wu & Palmer* similarity. In this preliminary study, we focus on the *Wu & Palmer* (WUP) similarity [11] while we aim to experiment with other metrics in the future. The WUP similarity between two synsets/words is defined as: $d(w_1, w_2) = 2 \times [depth\,(lcs(w_1, w_2))] \,/\, [depth(w_1) + depth(w_2)]$, where $lcs(.)$ is the *Least Common Subsumer* (LCS), also called the most specific ancestor node, and $depth(.)$ is the depth of a given node in the WordNet graph.

Multi-objective Optimization. Our approach uses the WordNet distance as an additional heuristic/objective to guide the search for optimal test case ordering rather than as an alternative to existing black-box heuristics. Given a test suite $T = \{t_1, \ldots, t_n\}$, we consider the following three objectives to optimize:

$$min \left[f_1(T) = \sum_{i=1}^{n} \frac{cost(t_i)}{i},\ f_2(T) = -\sum_{i=2}^{n} \frac{\mathrm{WD}(t_i, t_{i-1})}{i},\ f_3(T) = -\sum_{i=2}^{n} \frac{cosine(t_i, t_{i-1})}{i} \right] \tag{2}$$

f_1 measures the contribution of each test case t_i ($cost(t_i)$) to the cumulative execution cost divided by its position i in the test case ordering. It corresponds to the traditional cost-based objective used in the literature [2] for test case prioritization. f_2 measures the contribution of each test case t_i to the cumulative

diversity divided by its position i in the test case ordering. $WD(t_i, t_{i-1})$ denotes the WordNet distance between a test t_i and its predecessor t_{i-1} in the order. Finally, f_3 considers the traditional cosine distance between two consecutive test cases. These three objectives promote solutions where the least expensive or the more diverse test cases are prioritized first (i.e., executed earlier).

Finding optimal solutions for problems with multiple criteria requires trade-off analysis. Given the conflicting nature of our two objectives[1], it is not possible to obtain one single solution optimal for all objectives. Hence, we are interested in finding the set of solutions that are optimal trade-offs between the three objectives. To this aim, we use NSGA-II [5] as it provides well-distributed Pareto fronts and performs best when dealing with two or three search objectives [5].

3 Preliminary Study

The goal of our preliminary study is to answer the following research question:

RQ1: *To what extent does the use of WordNet improve the effectiveness of diversity-based test case prioritization?*

We compare `TestScheduler` against a baseline that prioritizes the test cases without using WordNet data. The baseline is set up as follows: it uses NSGA-II [2,12] and prioritizes the test cases based on (1) past execution cost and (2) cosine distance between the test cases. We assess the *effectiveness* of the prioritized tests in terms of fault detection capability, *i.e.,* detecting faults earlier when executing the test suite with a given test order.

Benchmark. For our preliminary study, we consider two open-source projects: `Javaparser` and `Apache commons lang`. The former is a Java library for parsing, analyzing, and manipulating Java code and it is publicly available on `GitHub`[2] (containing 2,864 test cases). For test execution cost (f_1 in Eq. 2), we collected the execution times available in the past build logs on the `GitHub` repository of the project. For the fault detection capability, we analyzed the *git* history and identified failing builds due to test failures and later fixed by developers via source code changes (patches). We consider only faults (7 in total) for which the developers provided a patch, and their commits do not include unrelated changes (e.g., documentation and refactoring).

The second project, i.e., `Apache commons lang`, is a library with a set of utility functions for the Java programming language (e.g., numerical function and string manipulation). We have selected this project since it is part of the Defects4J dataset [6]. The project has 2,223 test cases written in *JUnit* and has 65 isolated faults. The dataset also provides, for each isolated bug, the list of failing (fault-revealing) test cases. As for the execution cost, we run the test suites in a dedicated `Docker` container set up with the correct JVM version and

[1] Diverse tests are not necessarily the least expensive to run.

[2] https://github.com/javaparser/javaparser.

Table 1. Median $APFD_c$ values (with IQR) achieved with the WordNet Similarity vs. the traditional cosine distance. Best performance is highlighted in grey color.

Project Name	**Baseline**	`TestScheduler`	p-value	$\hat{A}_{12}$
`Javaparser`	0.8834 (0.0415)	0.9139 (0.0390)	0.004	0.7600 (large)
`Apache commons lang`	0.5230 (0.0335)	0.5778 (0.0455)	<0.001	0.8575 (large)

the required dependencies. We run each test case 10 times, and consider the median execution time as the (past) execution cost.

Experimental Setup. We run the `TestScheduler` (with WordNet) and the baseline (without WordNet) 20 times (each) to address their randomized nature. Both approaches rely on NSGA-II and are configured with the same default parameter values: (1) population size of 200 randomly sampled permutations; (2) 400 generations; (3) *partially-mapped crossover* (PMX) [2] with the probability μ_p=0.; (4) hybrid mutation operator [2] that combines three different mutation operators, namely *swap*, *insert*, and *invert*; (5) mutation probability of $\mu_c 0.1/\mid T \mid$ with T being the test suite to prioritize; (6) *tournament selection* with tournament size $k = 2$. Since the two approaches differ in the number of objectives they optimize for, we could not compare them based on traditional metrics, e.g., HV and IGD. Among the non-dominated solutions generated at the end of each search run, we selected the extreme (corner) points that achieved the lowest past execution cost. We have tried other solutions from the generated fronts (e.g., the knee or elbow solutions [2]), but they did not lead to consistent results. Therefore, we compared the two approaches based on the *cost-cognizant* variant of the *average percentage of faults detected* ($APFD_c$) [2,12] for the corner points described above. $APFD_c$ is a well-established metric in TCP, and it reflects practitioners' needs, interested in maximizing the number of detected regression faults at the same level of test execution cost. To assess the significance of the differences between `TestScheduler` and the baseline, we use the Wilcoxon rank-sum test with the α=0.05. We further complement the test for significance with the Vargha-Delaney statistic ($\hat{A}_{12}$).

Preliminary Results. Table 1 reports the median and the interquartile range (IQR) of the $APFD_c$ values achieved by `TestScheduler` and the baseline over 20 independent runs. We observe that `TestScheduler` achieves a larger (better) $APFD_c$ value than the baseline for both projects. For `javaparser`, `TestScheduler` leads to a median increase in the $APFD_c$ value of 3.05%; for `Apache commons lang`, the use of Wordnet leads to a median increase in $APFD_c$ of 5.48%. It is also worth noticing that simply relying on the cosine distance (the baselines) leads to $APFD_c$ values slightly above (very close) to what a random prioritization would do (i.e., $APFD_c$=0.5). From a statistical standpoint, the differences are all statistically significant according to the Wilcoxon rank-sum test

(all p-values<0.01) and with a *large* effect size according to the Vargha-Delaney statistic ($\hat{A}_{12}$=0.76 and $\hat{A}_{12}$=0.86 for `javaparser` and `Apache commons lang`, respectively). These preliminary results confirm our intuition/hypothesis that the use of WordNet can improve the fault detection capability.

Threats to Validity. The main threats to external validity regard the generalizability of our results. In this preliminary study, we have considered only two open-source projects. However, `javaparser` and `Apache common langs` are well-known and widely used projects in the software engineering community. The latter project is also part of the Defects4J dataset, which is a well-known benchmark for software testing research. Replicating our results with more projects and different application domains is part of our future plan.

4 Conclusion and Future Work

Diversity-based heuristics are widely used in TCP. In this paper, we propose to use WordNet to complement existing diversity metrics based on IR. We implemented a proof-concept in `TestScheduler`, and demonstrated the feasibility and usefulness of WordNet in a preliminary study. In the future, we plan to extend our study to more projects and different application domains. We also plan to investigate multiple distances for the WordNet taxonomy, such as the Leacock-Chodorow distance, and the Lin distance. Finally, we have considered only NSGA-II as the multi-objective algorithm. We plan to investigate other algorithms, such as AGE-MOEA [9] and NSGA-III.

Acknowledgements. This work has been partially supported by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 957254, project COSMOS [3].

References

1. Baeza-Yates, R., Ribeiro-Neto, B., et al.: Modern information retrieval, vol. 463. ACM press New York (1999)
2. Birchler, C., Khatiri, S., Derakhshanfar, P., Panichella, S., Panichella, A.: Single and multi-objective test cases prioritization for self-driving cars in virtual environments. ACM Trans. Softw. Eng. Methodol. (TOSEM) (2022)
3. COSMOS: Devops for complex cyber-physical systems. https://www.cosmos-devops.org (2021)
4. De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., Panichella, S.: Applying a smoothing filter to improve IR-based traceability recovery processes: an empirical investigation. Inform. Softw. Technol. (IST) **55**(4), 741–754 (2013)
5. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. on evol. Comput. **6**(2), 182–197 (2002)
6. Just, R., Jalali, D., Ernst, M.D.: Defects4J: a database of existing faults to enable controlled testing studies for Java programs. In: International Symposium on Software Testing and Analysis, pp. 437–440. ISSTA, ACM (2014)

7. Miller, G.A.: Wordnet: a lexical database for English. Commun. ACM **38**(11), 39–41 (1995)
8. Nguyen, C.D., Marchetto, A., Tonella, P.: Test case prioritization for audit testing of evolving web services using information retrieval techniques. In: International Conference on Web Services, pp. 636–643. IEEE (2011)
9. Panichella, A.: An adaptive evolutionary algorithm based on non-euclidean geometry for many-objective optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 595–603 (2019)
10. Peng, Q., Shi, A., Zhang, L.: Empirically revisiting and enhancing IR-based test-case prioritization. In: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 324–336 (2020)
11. Wu, Z., Palmer, M.: Verb semantics and lexical selection. arXiv preprint cmp-lg/9406033 (1994)
12. Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: a survey. Softw. Testing, Verification Reliab. **22**(2), 67–120 (2012)