SWAY for Decision Space of Permutations with Case Study on Test Case Prioritisation

Junghyun Lee, Chani Jung, Yoo Hwa Park, and Dongmin Lee

Abstract—The abstract goes here.

Index Terms—TCP, SWAY, Permutation metric, Permutohedron, Siemens programs.

1 Introduction

SOFTWARE engineers are faced with more and more problems with the growth of the software industry. Among many software engineering tasks, one of the most important problems is the test case prioritisation (TCP). Simply speaking, the goal of TCP is to find some optimal ordering of the test cases such that "early fault detection " is maximized. For this to happen, test cases that are more beneficial should be ordered to come before the ones that are less beneficial, so that fault detection occurs at an early stage of testing.

There were many attempts to solve TCP in prior studies, the most notable ones being greedy and search-based algorithms. However, in this paper, TCP is addressed with SWAY, a baseline optimizer for SBSE problems. As this problem is addressed with a baseline optimizer, the goal of this paper is to prove that SWAY can produce comparable results to the other state-of-the-art techniques, especially the search-based methods.

In this paper, SWAY and TCP are introduced to establish the background knowledge needed as a basis for our study. Also, a brief overview of the existing algorithms and their limitations are discussed. Then, our formulation of the method for applying SWAY to TCP is shown in detail, including embedding of the search space and adjustment of BETTER function. Finally, experiment design and results are shown, with thorough evaluation of the results.

The contributions of this work is as follows:

1) We present a novel embedding scheme for software engineering problem whose decision space is that of permutations, and show that SWAY can be directly used without any significant modifications.
2) To show that our algorithm performs competitively (and sometimes even better), SWAY is applied to Test Case Prioritisation. It is compared with additional greedy algorithm, which was recommended to be used in [1] due to its high performance.
3) Contrary to the promising performance of SWAY for TCP, by comparing the results with a completely random algorithm, we find that the Siemens programs are not very suitable to be used as benchmarks.

- *All authors contributed equally to this work*
- *Corresponding authors: Junghyun Lee, jh_lee00@kaist.ac.kr; Chani Jung, 1016chani@kaist.ac.kr; Yoo Hwa Park, 16ypark@kaist.ac.kr; Dongmin Lee, theresaldm@kaist.ac.kr*

Manuscript received December 21, 2020.

2 Background

2.1 SWAY

SWAY [2] is an alternative to multi-objective evolutionary algorithms(MOEA), used mainly for problems with conflicting objectives. SWAY recursively clusters possible candidates *in the decision space* until it is left with the superior cluster. In short, SWAY is about generating large population of candidates, clustering them and picking a representative from each cluster, evaluating only the representatives, and discarding any candidates that are embedded close to the poorly performing one. This results only $O(\log N)$ computation of the objective functions, which is very desirable for cases when the computation of objectives is expensive.

A *baseline* optimizer is an optimizer that is simple, widely and publicly available, fast, offer comparable performance to the SOTA methods, and inexpensive in terms of resource usage. (Such guidelines are due to [3]) It is beneficial to have a baseline optimizer for an optimization problem because it provides a floor performance values to it. This helps the developers to rule the optimizers with lower performance out. As argued in [2], SWAY satisfies all such criteria i.e. SWAY is a baseline optimizer for multi-objective search-based software engineering problems. Thus, SWAY can serve as a quick standard for comparing preliminary results, and can also serve as a guideline for the subsequent trials.

2.2 TCP

Let us first recall the rigorous definition of TCP [4]:

Definition 2.1. *Test Case Prioritisation (TCP)*

Input: *T, a test suite, PT, the set of permutations of T, and f, a function from PT to the real numbers.*

Problem: *Find* $T' \in PT$ *such that* $(\forall T'')\ (T'' \in PT)\ (T'' \neq T')\ [f(T') \geq f(T'')]$.

There are two main approaches to TCP: coverage-based and diversity-based [5]. Coverage-based TCP makes use of coverage information of each test case to produce an ordering, while diversity-based TCP makes use of how diverse each of the test cases are in order to select the most

diverse subset of the test cases. In this work, we focus on the coverage-based approach

3 Existing Approaches for TCP

3.1 Additional Greedy Algorithms

Additional greedy algorithm is widely utilized for TCP. It starts from an empty solution, and then for each iteration, the test case which gives the maximal coverage gain to the partial solution is added.

However, due to the structure of selecting locally optimal solution at each iteration, additional greedy algorithm does not always produce the optimal solution for TCP. There exist the cases that the algorithm falls into sub-optimal local minima, which is shown using a small toy example in [1]. In the example, we can see that starting with the maximally covering test case does not always achieve the earliest full coverage of the program.

3.2 Search-Based Algorithms

There are also search-based methods to solve TCP. The most frequently used are 2-Optimal, Hill Climbing, and Genetic Algorithm. Refer to [1] on how the required operations (e.g. fitness function for genetic algorithm) are chosen. However, these approaches have expensive time cost because they require large number of evaluations of the candidates to reach to the solution. They also have threats to fall into local minima, but not as much since these aren't deterministic like greedy algorithm.

3.3 What do we compare against?

According to [1], there is no significant difference in performance between search-based and greedy algorithm. Therefore, we chose additional greedy algorithm, which is cheaper to implement and execute, as the one to compare the performance of our algorithm with.

4 SWAY for TCP

4.1 Background

Let $\mathcal{T} = \{t_1, \ldots, t_n\}$ be the set of all test cases. Recall that SWAY clusters the candidates through their *decisions* rather than *objectives*. Thus in order to apply SWAY to TCP, one needs to model TCP as a modeling problem of converting *decisions* d into *objective* scores o i.e.

$$o = TCP(d)$$

Noting that any such objective scores can be obtained by executing the whole test suite in the given order, it is natural to think of d as the **ordering of** $\mathcal{T}$, which will be rigorously handled by using permutation as described in the next section.

4.2 Mathematical preliminaries

This section provides a brief overview of the necessary mathematical concepts. Let us start with a basic definition:

Definition 4.1. *A* **permutation** *of* $[n] := \{1, 2, \ldots, n\}$ *is a bijection from* $[n]$ *to itself. Denote* S_n *as the set of all possible permutations of* $[n]$.[1] *Espeically, let* $i = (1, \ldots, n) \in S_n$ *be the identity permutation.*

Permutation is one of the most fundamental combinatorial objects of immense range of practical applications, from computational biology to software engineering. In our problem, assuming that we have n different test cases to order, it can be observed that *each permutation of* $[n]$ *corresponds to a unique ordering of the given test suite*, giving us a direct problem-specific interpretation.

SWAY depends on the assumption that there exists a close association between the decision and objective spaces. In our case, since we are considering coverage-based TCP, it is desirable to have the following relation: if two test suites are "close" to each other, then their "degree" of coverage should be similar.

This motivates the need to compare two permutations i.e. compute their "distance". Unlike $\mathbb{R}^p$, which has a natural metric endowed from its p-norm (l_p-metric), there isn't a "natural" way of computing the distance between two permutations.

Let us first introduce some concepts [6]:

Definition 4.2. *Given a connected*[2] *graph* $G = (V, E)$*, its* **path metric** *is a metric on* V*, defined as the length(# of edges) of a shortest path connecting two given vertices* x *and* y *from* V*.*

Definition 4.3. *Given a finite set* X *and a finite set* $\mathcal{O}$ *of unary editing operations on* X*, the* **editing metric** *on* X *is the path metric of the graph* $G(X, \mathcal{O})$ *whose vertices are* X *and whose edge set is* $\{\{x, y\} \mid \exists o \in \mathcal{O}\ y = o(x)\}$*.*

If $X = S_n$*, then it is also called* **permutation metric***.*

(Metric is a distance function from $X \times X$ to $\mathbb{R}_{\geq 0}$ satisfying three axioms. Refer to [7] for a more detailed discussion on the topic.)

4.3 Intuition

Unlike continuous space, in which one needs to move in a continuous manner to go from one point to another, our space S_n is discrete. The most natural way of moving from one permutation to another is by *switching two elements*, formalized by the following notion:

Definition 4.4. $\pi = (\pi_1, \ldots, \pi_n) \in S_n$ *is called a* **transposition** *if it switches two elements, only i.e. it is a cycle of length* 2 *i.e. there exists some* $i \neq j \in [n]$ *such that* $\pi_i = j, \pi_j = i$ *and* $\pi_k = k$ *for all* $k \in [n] \setminus \{i, j\}$*.*

One should observe, however, that our problem is heavily dependent on the relative ordering of the test cases. For example, let $\pi = (1\ 2\ 3\ 4)$. Both $\pi' = (1\ 3\ 2\ 4)$ and $\pi'' = (4\ 2\ 3\ 1)$ differ by a single transposition from π, but

1. Note that S_n and usual function composition operation is precisely the symmetric group of order n.

2. A graph $G = (V, E)$ is connected if for every $x, y \in V$, there exists a path from x to y

in terms of the *"degree" of early coverage*, they may differ significantly. In particular, it is most likely that π and π' have similar "degree" of early coverage, while the opposite for π and π''. Thus we resort ourselves to the following special type of transposition:

Definition 4.5. *$\pi = (\pi_1, \dots, \pi_n) \in S_n$ is called a* **swap** *if it is a transposition that switches two adjacent elements i.e. there exists some $i \in [n]$ such that $\pi_i = i+1, \pi_{i+1} = i$ and $\pi_k = k$ for all $k \in [n] \setminus \{i, i+1\}$.*

Again, the reason for using this is because we are interested more in seeing how "local" "locality" (EXAMPLE: general transposition may change the objective a lot since the objective is dependent on the **relative ordering** of the elements i.e. as the distance between two switched test cases increase, the change in relative orderings of the elements becomes more severe)

Since our main goal is to somehow cluster S_n, based on the previous discussion, it is natural to consider the following permutation metric:

Definition 4.6. *The* **swap distance** *(also known as* **Kendall τ distance** *in statistical ranking theory) of $\pi, \pi' \in S_n$, denoted as $d_K(\pi, \pi')$, is the editing metric on S_n with $\mathcal{O}$ being the set of all possible swaps i.e. it is the minimum number of swaps required to go from π to π'.*

(Verifying that the above-defined swap distance is indeed a metric is left to the readers as a simple exercise)

Lastly, the following lemma provides a very intuitive way of computing the swap distance:

Lemma 4.1. *Given $\pi, \pi' \in S_n$, $d_K(\pi, \pi')$ is precisely the number of relative inversions between them i.e. number of pairs $(i, j), 1 \le i < j \le n$ with $(\pi_i - \pi_j)(\pi'_i - \pi'_j) < 0$.*

Now our goal is to find an appropriate embedding scheme that preserves the following property: *two permutations are close together if they differ by small number of swaps, and vice versa*. The next two sections show two different such embeddings, along with short mathematical justifications.

4.4 Embedding Scheme

4.4.1 Overview

Let us recall some concepts from combinatorics and polytope theory:

Definition 4.7. *For fixed n, the* **permutahedron**, *denoted as Π_{n-1}, is defined as the convex hull of the set $S = \{(\pi(1), \dots, \pi(n)) \mid \pi \in S_n\}$.*

Permutahedron, first introduced in [8], has been a subject of intensive study in the field of not only combinatorics, but also in other fields such as bandit optimization [9], . We shall look at two important properties of Π_{n-1} and their implications in our current problem. Refer to [10], [11] for the full proofs and more detailed discussions on related topics.

Lemma 4.2. *Π_{n-1} is a simple polytope of dimension $n-1$, with $n!$ vertices given as S.*

This shows that directly embedding S onto $\mathbb{R}^n$.

Lemma 4.3. *Π_{n-1} is a geometric realization of weak Bruhat order on S_n i.e. two vertices of Π_{n-1} are adjacent iff they differ by a swap.*

Above lemma has the important implication that *such simple and intuitive geometric realization(embedding) of S_n onto $\mathbb{R}^{n-1}$ (with the usual Euclidean metric) has a very mathematically desirable structure*. In other words, this motivates for us to **directly use continuous SWAY [2]**!

4.4.2 Mathematical Justification

However, it is not clear at first of how the $l2$-norm of two permutations is related to their swap distance. It may be that even though two permutations are far apart in $\mathbb{R}^{n-1}$ in $l2$-distance, they may be similar in swap distance or vice versa. In this subsection, we provide a mathematical justification that such case is not possible, in general.

To see this, we need some concepts from statistical ranking theory:

Definition 4.8. Spearman ρ distance *of $\pi, \pi' \in S_n$, denoted as $d_S(\pi, \pi')$, is precisely the Euclidean distance between π and π', considering them as vectors (vertices of Π_{n-1} in $\mathbb{R}^n$)*

Definition 4.9. Daniels-Guilbaud semi-metric[3] *(abbreviated as DG-distance) of $\pi, \pi' \in S_n$, denoted as $d_G(\pi, \pi')$, is defined as the number of triples $(i, j, k),\ 1 \le i < j < k \le n$ such that (π_i, π_j, π_k) is not a cyclic shift of (π'_i, π'_j, π'_k).*

Definition 4.10. *$\pi, \pi' \in S_n$ are said to have a* **circular agreement** *on $\{x, y, z\} \subset [n]$ if (π_x, π_y, π_z) can be obtained via a circular permutation of (π'_x, π'_y, π'_z).*

The following results from [12] provide a key theoretical characterization of our approach[4]:

Theorem 4.4 (Monjardet, 1998)**.**

$$d_S^2(\pi, \pi') = n d_K(\pi, \pi') - d_G(\pi, \pi') \quad \forall \pi, \pi' \in S_n$$

Lemma 4.5 (Monjardet, 1998)**.**

$$d_G(\pi, \pi') + a_G(\pi, \pi') = \binom{n}{3}$$

Here, $a_G(\pi, \pi')$ is defined as the number of triplets $\{x, y, z\} \in [n]$ such that π and π' have a circular agreement on. (Basically, it can be thought of as a "dual" to d_G)

Writing it in another way,

$$d_S^2(\pi, \pi') = n d_K(\pi, \pi') + a_G(\pi, \pi') - \binom{n}{3}$$

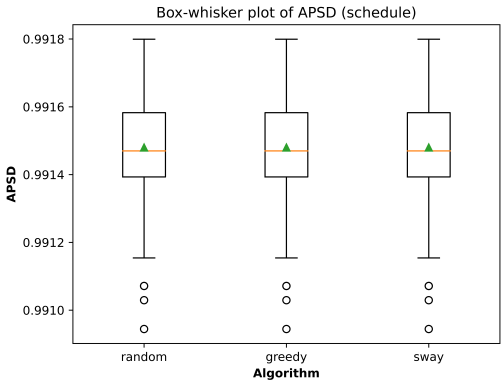
One can thus say that π, π' are far apart in $\mathbb{R}^n$ if

1) $d_K(\pi, \pi')$ is large i.e. they differ by a lot of swaps.
2) $a_G(\pi, \pi')$ is large i.e. they are quite mixed up.

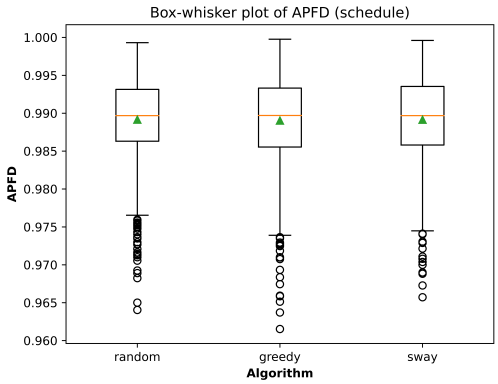
Based on these mathematically rigorous results, we argue that our embedding, up to some level of distortion,

3. Semi-metric is a generalized metric that doesn't satisfy the triangle inequality.

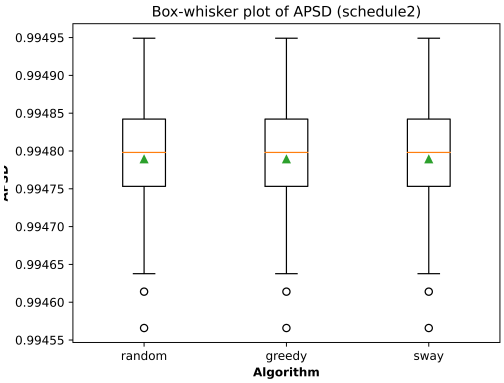
4. In our final presentation video, we claimed that there was an elegant statistical characterization of $d_G(\cdot, \cdot)$ and deferred the proof to here. However, after a careful scrutinization, we realized that there was a significant flaw in the proof, thus forcing us to remove the "result". Still, we provide an equally elegant theoretical explanation for our embedding, although only based on existing results.



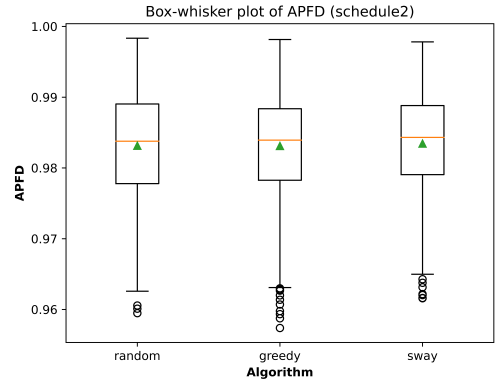
(a) **schedule**, APSD



(b) **schedule**, APFD



(c) **schedule2**, APSD



(d) **schedule2**, APFD

Fig. 1: Comparison of APSD, APFD over three algorithms for **schedule, schedule2**. (Beware of the scale! For instance, (c) and (d) have significantly different scales for the y-axis!)

accurately model the swap distance of two permutations, and thus it is okay to use the continuous version of SPLIT for our problem.

4.5 Initial population

For small n, using all of S_n is not so much of a problem. However, it becomes a big problem when n is big, especially in many of the industrial cases. Using all of S_n for initialization requires $n!$ points, multiplied by the space complexity required for used embedding scheme.

Thus we propose generating 2^k random permutations from S_n. Such sampling is done using Fisher-Yates shuffle, which outputs uniformly distributed permutations [](**CITATION**). We use Python's RANDOM.SHUFFLE function, which makes use of the Fisher-Yates algorithm.

4.6 BETTER function

Using the already-known execution information of each test case, we've used APSD [4] as our BETTER function, given as

$$APSD(T) = 1 - \frac{TS_1 + \cdots + TS_m}{nm} + \frac{1}{2n} \tag{1}$$

where TS_i is the index of the first test case that covers statement i, n is the number of test cases in the test suite, and m is the number of statements in the program. Such choice is natural since our current approach is coverage-based, and APSD is the most widely used function for measuring the early coverage percentage.

5 EXPERIMENTS

5.1 Benchmarks

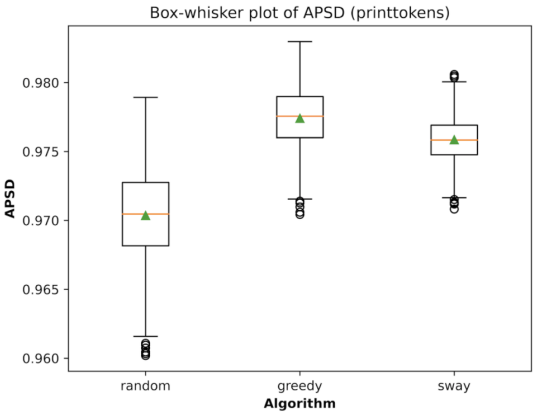
Software-artifact Infrastructure Repository (SIR) [13] is an infrastructure, created to support controlled experimentation with testing techniques. It is home to many software-related artifacts that support rigorous controlled experimentation with program analysis and software testing techniques.

We consider 6 Siemens programs, introduced by the Siemens Corporate Research for a study of the fault detection capabilities of control-flow and data-flow coverage criteria [14]. Those programs were developed to study the fault detection of the code. This programs perform a variety of tasks (also referred to as programs): **schedule, schedule2, printtokens, printtokens2, tcas, totinfo**. For each programs, it contains large pool of test cases.

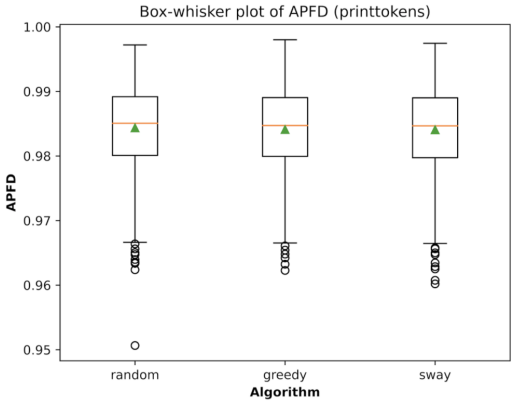
Below is a brief description of each program:

1) **schedule, schedule2**: priority schedulers.
2) **printtokens, printtokens2**: lexical analysers
3) **tcas**: an aircraft collision avoidance system.
4) **totinfo**: computes statistics of given input data.

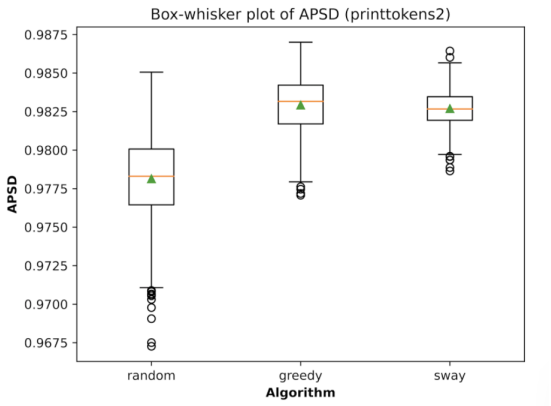
Refer to [14] for a detailed explanation on how the programs were seeded with faults, and how the test suites



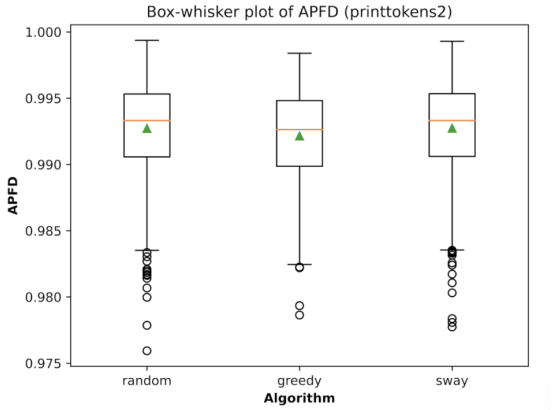
(a) **printtokens**, APSD



(b) **printtokens**, APFD



(c) **printtokens2**, APSD



(d) **printtokens2**, APFD

Fig. 2: Comparison of APSD, APFD over three algorithms for **printtokens, printtokens2**.

were made. For our experiment, we considered the 1000 test suites in the folder *tesplans-bigcov*; those of which are suspected to have large sizes with full coverage.

5.2 Research Questions

To explore our approach of applying SWAY to TCP, we organize our exploration around the following research questions (RQ):

1) **(RQ1)** How well SWAY performs compared to the state-of-the-art method for TCP?
2) **(RQ2)** How sensitive is SWAY to the initial population? (which is practically the only hyperparameter)

As mentioned in Section 3, we only consider additional greedy algorithm as our competitor since it is simple to implement, and it shows comparable performance with the other search-based algorithms. In addition, for sanity check, we've also considered random algorithm, which just outputs a random permutation. For each program, a boxplot of APSD and APFD is plotted for each algorithm, where all 1000 test suites were considered.

5.3 Performance Measures

We had used two metrics for the performance measure of SWAY and the compared existing methods.

The first metric is Average Percentage of Statement Detection(APSD). It measures how early the test suite can cover the whole statements of the given program. Since SWAY took coverage-based approach for TCP, we measured and compared APSD for the results of each method to see if SWAY had worked properly as it was expected. Its formula is given in Eq. 1.

The second metric we had used is Average Percentage of Fault Detection(APFD), which is a widely used performance metric for TCP problems. It measures how early the test suite can detect the faults of the whole program, which accords with the goal of TCP. APFD is given as

$$APFD(T) = 1 - \frac{TF_1 + \cdots + TF_m}{nm} + \frac{1}{2n}$$

where TF_i is the index of the first test case that detects fault i, n is the number of test cases in the test suite, and m is the number of faults in the program.

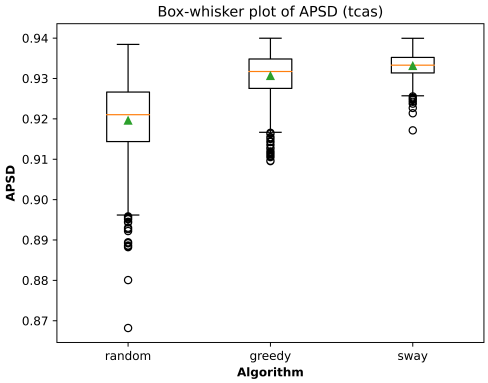
6 Results

All our codes are available in our Github repository[5].

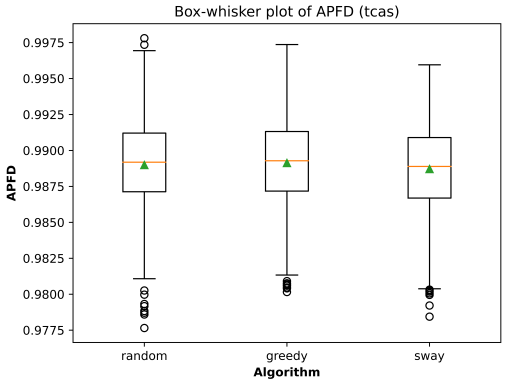
6.1 RQ1: How well does our approach perform?

Figures 1, 2, 3 show the resulting box-plots of all the programs considered. Green triangle is the mean of the metric considered. For simplicity, additional greedy algorithm is labeled as "greedy".

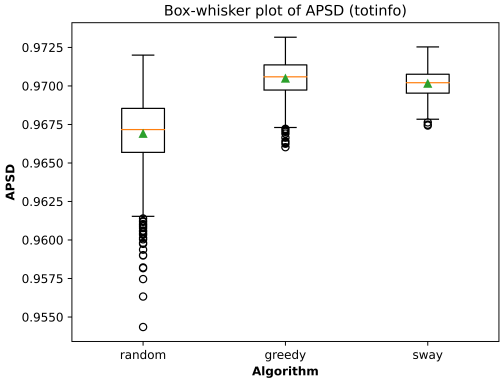
5. https://github.com/Dongmin1215/CS454_Team5



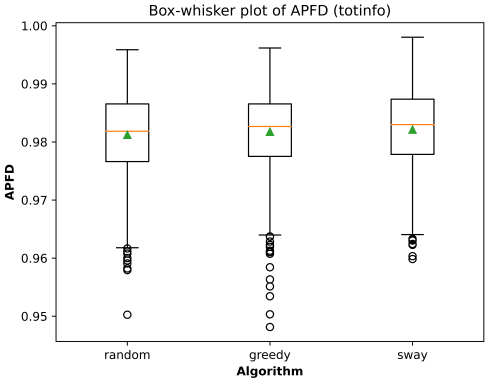
(a) **tcas**, APSD



(b) **tcas**, APFD

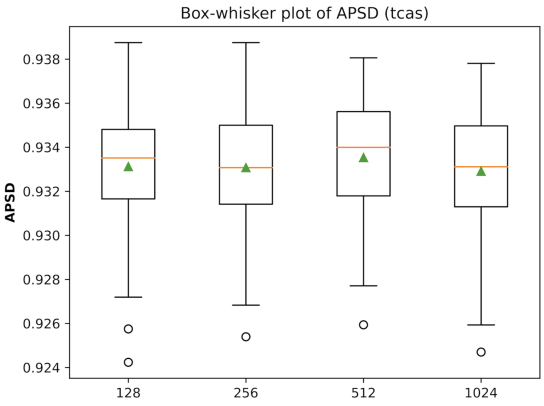


(c) **totinfo**, APSD

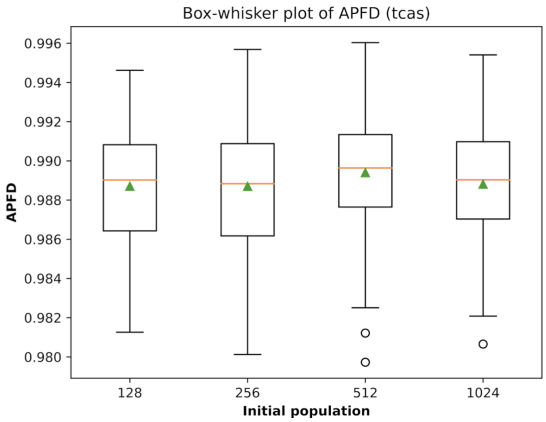


(d) **totinfo**, APFD

Fig. 3: Comparison of APSD, APFD over three algorithms for **tcas, totinfo**.



(a) APSD



(b) APFD

Fig. 4: Effect of hyperparameters(initial population) on APSD and APFD, for tcas.

As for APSD, all three algorithms considered did not show a significant difference among all the programs considered. Although there are some programs in which SWAY was outperformed by the additional greedy algorithm, it can be seen that SWAY performs comparably with the additional greedy algorithm. Note that in general, observing the number of outliers and the location of quartiles, it can be claimed that SWAY has less variance than random and additional greedy algorithm, showing that SWAY-based approach is more robust to different test suites. Similar trend is observed for APFD, although not as clear as APSD since we're directly optimizing over APSD.

As an additional observation, note that there are some programs in which a uniformly randomly chosen permutation performs comparably with both additional greedy algorithm and SWAY (e.g. **schedule**). We suspect that this is due to the excessively simple structure of the faults seeded in the programs, and thus the difference between different test cases orderings is very subtle.

6.2 RQ2: Sensitivity of hyperparameters

For the sake of simple exposition, we did this experiment with the **tcas** program, only. However, we suspect that similar results will show up for other Siemens programs. We considered the initial population of $\{128, 256, 512, 1024\}$. As one can observe in Figure 4, the results do not show a big difference. This is a rather surprising, yet desirable, result since it implies that our algorithm is very robust to the initial population (despite the intuition that more initial population leads to larger searches and thus better performance). However, since this is only for the Siemens programs, future research should explore whether this holds for other (more complex) programs, such as **space** [15].

7 Conclusions and Future Work

In this paper, we propose a novel way of applying SWAY, a baseline optimizer for multi-objective software engineering problems, to TCP (and possibly other problems whose decision space is the space of permutations). Specifically, we adopted the embedding of the permutations into the Euclidean space, and set the BETTER function as APSD which let the algorithm take coverage-based approach of TCP. We provide rigorous contextual evidence from well-established fields such as combinatorics, polytope theory, and statistical ranking theory to show that our embedding scheme is suitable for the framework of SWAY.

In RQ1, we verified our algorithm by comparing it with random permutations and the widely used additional greedy algorithm, for the Siemens programs. Both in terms of APSD and APFD, SWAY performed comparably with additional greedy algorithm. For some programs, SWAY is more robust in terms of different test suites. In RQ2, we observed that initial population size didn't have much impact on the performance of SWAY for **tcas** program, although we suspect that this would be the case for other programs as well. Additionally, we observe that random permutation is at par with additional greedy algorithm and SWAY for some of the Siemens programs, hinting that they might not be suitable for future research on test case prioritisation.

One immediate direction for extending this work is to try another embedding, as described in Appendix A. This embedding exactly preserves the swap distance between two permutations, without any distortion, but at the cost of higher computational complexity. Due to time constraint, we couldn't implement this strategy, and so it would be interesting to see if this strategy is effective.

Another is to consider some variants of TCP:

1) Since SWAY was proposed as an alternative to multi-objective problem, it would be interesting to use the same embedding for more complex variants of TCP such as time-constrained prioritisation [16], [17].
2) Here, we've only considered the approach of considering the permutation as the decision space i.e. each embedded point is a distinct test suite. It would be interesting to see if it is possible (and if it is maybe better) to embed each test case as a point. This idea may be lead to a new algorithm when considering a diversity-based approach to TCP.
3) TCP considered here is called *white-box execution-based prioritisation*) [18] because we assumed that we have access to the source code of the SUT, and that we could execute the test cases. It would be interesting to see how to apply SWAY in the opposite situation, often called *black-box static prioritisation*. [18], [19], [20] tackled this problem by using string embedding, of which the future algorithm may be benefited from.

Although our work is only about TCP, it is straightforward to try applying our algorithm to other SBSE problems with decision space of permutations. Some examples are:

- Traveling salesman problem(TSP)
- Permutation flow shop scheduling problem
- Quadratic assignment
- Linear ordering

Acknowledgments

The authors would like to thank Prof. Shin Yoo and Seungmin Lee of COINSE Lab (KAIST) for their insightful advices and guidances.

References

[1] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Transactions on Software Engineering*, vol. 33, no. 4, pp. 225–237, 2007.

[2] J. Chen, V. Nair, R. Krishna, and T. Menzies, ""Sampling" as a Baseline Optimizer for Search-Based Software Engineering," *IEEE Transactions on Software Engineering*, vol. 45, no. 6, pp. 597–614, 2019.

[3] P. A. Whigham, C. A. Owen, and S. G. Macdonell, "A Baseline Model for Software Effort Estimation," vol. 24, no. 3, 2015. [Online]. Available: https://doi.org/10.1145/2738037

[4] G. Rothermel, R. J. Untch, and C. Chu, "Prioritizing test cases for regression testing," vol. 27, no. 10, 2001. [Online]. Available: https://doi.org/10.1109/32.962562

[5] D. Mondal, H. Hemmati, and S. Durocher, "Exploring test suite diversification and code coverage in multi-objective test case selection," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, 2015, pp. 1–10.

[6] M. M. Deza and E. Deza, *Encyclopedia of Distances*, 4th ed. Springer-Verlag Berlin Heidelberg, 2018.

[7] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed. McGraw-Hill, 1976.

[8] P. H. Schoute, "Analytic treatment of the polytopes regularly derived from the regular polytopes," *Verhandelingen der Koninklijke Akademie van Wetenschappen Te Amsterdam*, vol. 11, no. 3, pp. 370–381, 1911.
[9] N. Ailon, K. Hatano, and E. Takimoto, "Bandit online optimization over the permutahedron," in *Algorithmic Learning Theory*, P. Auer, A. Clark, T. Zeugmann, and S. Zilles, Eds. Cham: Springer International Publishing, 2014, pp. 215–229.
[10] G. M. Ziegler, *Lectures on Polytopes*, ser. Graduate Texts in Mathematics. Springer-Verlag New York, 2007, vol. 152.
[11] R. P. Stanley, *Enumerative Combinatorics: Volume 1*, 2nd ed., ser. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2012, vol. 49.
[12] B. Monjardet, "On the comparison of the spearman and kendall metrics between linear orders," *Discrete Mathematics*, vol. 192, no. 1, pp. 281 – 292, 1998.
[13] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, no. 4, p. 405–435, Oct. 2005. [Online]. Available: https://doi.org/10.1007/s10664-005-3861-2
[14] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria," in *Proceedings of 16th International Conference on Software Engineering*, 1994, pp. 191–200.
[15] F. I. Vokolos and P. G. Frankl, "Empirical evaluation of the textual differencing regression testing technique," in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, 1998, pp. 44–53.
[16] Jung-Min Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, 2002, pp. 119–129.
[17] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel, "The effects of time constraints on test case prioritization: A series of controlled experiments," *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 593–617, 2010.
[18] S. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein, "Static test case prioritization using topic models," *Empirical Software Engineering*, vol. 19, 02 2014.
[19] Y. Ledru, A. Petrenko, and S. Boroday, "Using string distances for test case prioritisation," in *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '09. USA: IEEE Computer Society, 2009, p. 510–514. [Online]. Available: https://doi.org/10.1109/ASE.2009.23
[20] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritising test cases with string distances," *Automated Software Engineering*, vol. 19, pp. 65–95, 03 2012.
[21] S. M. Moser and P.-N. Chen, *A Students Guide to Coding and Information Theory*. Cambridge University Press, 2012.
[22] G. Cormode, "Sequence distance embedding," Ph.D. dissertation, University of Warwick, 1 2003.

APPENDIX A
ANOTHER EMBEDDING

Let us recall an important concept from coding theory [21]:

Definition A.1. *Given any two vectors(codewords)* $x = (x_1, \ldots, x_n)$ *and* $y = (y_1, \ldots, y_n)$*, the* **Hamming distance** *between* x, y*, denoted as* $d_H(x, y)$*, is defined as*

$$d_H(x, y) := \sum_{i=1}^{n} \left[x_i \stackrel{?}{=} y_i \right]$$

where $\left[x_i \stackrel{?}{=} y_i \right]$ *is the indicator function of whether* x_i *is equal to* y_i*.*

(The definition naturally extends for two equal-sized matrices by considering their vectorizations.)

One can immediately see that if $x, y \in \{0, 1\}^n$, then $d_H(x, y)$ can be rewritten as $\sum_{i=1}^{n} |x_i - y_i|$ i.e. the *Jaccard distance* between x and y.

The following lemma from [22] is the key theoretical basis for this embedding:

Lemma A.1.

$$\forall \pi, \pi' \in S_n \quad d_K(\pi, \pi') = d_H\left(S(\pi), S(\pi')\right)$$

where $[S(\pi)]_{ij} := [i < j \wedge \pi^{-1}(i) < \pi^{-1}(j)]$ *and* $[\cdot]$ *is the indicator function for the given logical predicate.*

The significance of above lemma is that *such binary embedding scheme gives a distortion-free embedding of* S_n *onto* $\{0, 1\}^n$ *with Hamming distance as the endowed metric*[6] i.e. we can **directly use binary SWAY** as proposed in [2]! Moreover, note that in our binary case, the Hamming distance is equivalent to the Jaccard distance, providing further justification for the direct application of binary SWAY.

Lastly, noting that for all $\pi \in S_n$, $S(\pi)$ is anti-symmetric (if (i, j)-th component is 1, then (j, i)-th component is 0 and (i, i)-th component is 0), we can reduce the space complexity from n^2 to $\frac{n(n-1)}{2}$.

Note that in exchange for preserving the pairwise swap distances, the space complexity (and thus the expected computational complexity) is rather high. It takes $O(N_0 n^2)$ space complexity with $O(N_0 n)$ time complexity for embedding each permutation, where N_0 is the size of the initial population. Our embedding scheme as mentioned in Section 4.4 takes $O(N_0 n)$ space complexity with $O(N_0)$ time complexity for embedding each permutation, but at the cost of some distortion introduced. Thus there is a clear trade-off between the two embedding schemes, and it would be interesting to see if such trade-off extends to performance of the algorithm.

6. This space is known as the *Hamming space* in coding theory.