Preliminary Evaluation of SWAY in Permutation Decision Space via a Novel Euclidean Embedding

Junghyun Lee[⋆], Chani Jung[⋆], Yoo Hwa Park[⋆], Dongmin Lee[⋆], Juyeon Yoon, and Shin Yoo

KAIST, Daejeon, South Korea
{jh_lee00,1016chani,16ypark,theresaldm,juyeon.yoon,shin.yoo}@kaist.ac.kr

Abstract. The cost of a fitness evaluation is often cited as one of the weaknesses of Search-Based Software Engineering: to obtain a single final solution, a meta-heuristic search algorithm has to evaluate the fitness of many interim solutions. Recently, a sampling-based approach called SWAY has been introduced as a new baseline that can compete with state-of-the-art search algorithms with significantly fewer fitness evaluations. However, SWAY has been introduced and evaluated only in numeric and Boolean decision spaces. This paper extends SWAY to permutation decision space. We start by presenting the theoretical formulation of the permutation decision space and the distance function required by SWAY, and subsequently present a proof-of-concept study of Test Case Prioritisation (TCP) problem using our permutative SWAY. The results show that our embedding works well for permutative decision spaces, producing results that are comparable to those generated by the additional greedy algorithm, one of the most widely used algorithms for TCP.

Keywords: SWAY · Permutations · TCP.

1 Introduction

Search Based Software Engineering (SBSE) formulates software engineering problems as metaheuristic optimisations and applies various search techniques [10]. These search techniques typically navigate the decision (or solution) space guided by a fitness function that maps a solution in the decision space to one or more values in the fitness (or objective) space. The mapping is achieved by dynamically evaluating the decision against the actual problem instance and measuring the fitness. The dynamic and concrete nature of fitness evaluation allows us to search for properties that cannot be easily measured otherwise: for example, Search Based Software Testing has successfully applied to find test inputs that satisfy non-functional test requirements such as increased memory consumption [13] or worst case execution time [16].

⋆ Equal contribution

However, such dynamic and concrete nature of fitness evaluation is accompanied by the cost of the actual execution. Given the wide adoption of population based optimisation algorithms such as genetic algorithm, the cost of fitness evaluation poses a serious threat not only to practical applications of SBSE but also to research and experimental use of these techniques.

Recently, a new type of search algorithm based on random sampling, called SWAY, has been introduced [2]. Suppose we are optimising an objective variable by searching for a decision variable. The fitness evaluation can be formulated as:

$$o = fitness(d)$$

where $d \in \mathcal{D}$ is the decision variable and $o \in \mathcal{O}$ is the objective variable. Chen et al. empirically showed that, in many cases of SBSE formulations, there exists a close association between $\mathcal{D}$ and $\mathcal{O}$ [2]. SWAY exploits this by recursively clustering, and searching for, possible candidate solutions *in the decision space* rather than in the objective space. By doing so, it admits a logarithmic time complexity in terms of fitness evaluations. Such scalability, along with its simplicity, qualifies SWAY as a baseline optimizer [21]. However, the current form of SWAY is limited to decision spaces that are either numerical or Boolean. The aim of this paper is to extend SWAY to permutative decision space.

The technical contributions of this paper are as follows:

- We present a formulation of SWAY in permutative decision spaces. Instead of coarse-grained grouping followed by the use of binary SWAY (as suggested by authors of SWAY), we present a novel Euclidean embedding of permutations that can be used with continuous SWAY.[1]
- We conduct a proof-of-concept evaluation of SWAY in a permutative decision space with instances of Test Case Prioritisation problems. The results show the feasibility of our embedding.

The rest of the paper is organised as follows. Section 2 introduces the original SWAY, proposes our novel Euclidean embedding of permutative decision spaces, and introduces the adaptation of SWAY to the proposed embedding. Section 3 describes the settings of the case study of the application of SWAY to Test Case Prioritisation problem, whose results are presented in Section 4. Section 5 discusses the related work, and Section 6 concludes.

2 SWAY for Permutative Decision Spaces

This section first introduces the basic SWAY algorithm, and subsequently presents the design of our embedding scheme for permutation decision variables.

[1] Here, embedding for a permutative decision space simply refers to a mapping of each permutation to some Euclidean space that is structure-preserving (as in "node embedding" in machine learning). In the following sections, we shall describe which structure to preserve.

2.1 The Original SWAY

SWAY [2, 3] is an effective random sampling algorithm that can be used as a baseline for more sophisticated search algorithms. Algorithm 1 shows the pseudocode of the original, continuous SWAY. At its core, SWAY simply seeks to choose a cluster of solutions that are superior to others. If the clustering is performed based on the solution phenotype, SWAY would have to spend a lot of fitness evaluations. Instead, SWAY exploits the fact that, in many SBSE problems, there exists a close association between the genotype (i.e., decision) and the phenotype (i.e., objective) spaces [3], and recursively clusters the solutions in the genotype space using FastMap heuristic [9] (implemented in SPLIT), only evaluating the representatives of each cluster.

Algorithm 1: Continuous SWAY with its subroutine Split

```
Given: inner product $\langle \cdot, \cdot, \rangle$ and its induced norm $\langle \|\cdot\| \rangle$, objective
  computing function $obj : \mathcal{D} \to \mathcal{O}$, ordering on $\mathcal{O}$ $\preccurlyeq$
Hyperparameters: $enough$
Function contSWAY($candidates$):
    if $|candidates| < enough$ then
        return $candidates$
    else
        $[west, westItems], [east, eastItems] \longleftarrow$ Split($candidates$)
        $\Delta_1, \Delta_2 \longleftarrow \emptyset, \emptyset$
        if $obj(east) \preccurlyeq obj(west)$ then
            $\Delta_1 \longleftarrow$ contSWAY($westItems$)
        end
        if $obj(west) \preccurlyeq obj(east)$ then
            $\Delta_2 \longleftarrow$ contSWAY($eastItems$)
        end
        return $\Delta_1 \cup \Delta_2$
    end
End Function
Function Split($candidates$):
    $rand \longleftarrow$ randomly selected candidate from $candidates$
    $east \longleftarrow \text{argmax}_{x \in candidates} \|x - rand\|$
    $west \longleftarrow \text{argmax}_{x \in candidates} \|x - east\|$
    for $x \in candidates$ do
        $x_d \longleftarrow \frac{\langle east-west, x-west \rangle}{\|east-west\|}$
    end
    Sort candidates by $x_d$
    $eastItems \longleftarrow$ first half of $candidates$
    $westItems \longleftarrow$ second half of $candidates$
    return $[west, westItems], [east, eastItems]$
End Function
```

For problem with continuous numerical genotypes, the FastMap heuristic that is based on cosine rules and Euclidean distance works well (Line 22 of Algorithm 1); for binary decision spaces, SWAY adopts a radial coordinate system. Finally, for non-binary discrete decision spaces, Chen et al. propose coarse-grained binary groupings of such solutions fed into the binary version of SWAY [2, 3]. However, even with sufficient domain knowledge, certain decision spaces may not allow an easy and intuitive coarse-grained grouping that would enable us to use binary SWAY. For example, it is not clear what coarse-grained groupings can be used in a permutative decision space without knowing which ordering is better than others.

We propose a formulation of SWAY for permutative decision spaces that uses the continuous SWAY. Our intuition is that we can use the continuous SWAY as long as we can *embed* permutations into an alternative vector form in such a way that the Euclidean distance between embedding vectors is closely correlated with Kendall τ distance, i.e., the combinatorial distance between permutations (i.e., the number of pairwise disagreement between two permutations). Note that the use of continuous SWAY, which depends on the cosine rule and Euclidean space, forces us to use Euclidean embedding.

The remainder of this section investigates such an embedding of the set of all possible permutations, denoted as S_n. To the best of our knowledge, S_n cannot be endowed with an easy-to-be-implemented inner product, or even a well-defined one. Thus, we need to embed S_n into a simple inner vector space.

2.2 Preliminaries

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

Definition 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 $[n]$.*[2] *Especially, let $i = (1, \ldots, n) \in S_n$ be the identity permutation. Moreover, depending on the context, $\pi \in S_n$ may be regarded as a vector in $\mathbb{R}^n$.*

Unlike a p-dimensional space $\mathbb{R}^p$, which has a natural metric endowed from its Euclidean norm[3], defining the distance between two permutations in an analogous manner is not trivial. First, let us start with the "natural" definition of metric on S_n (much discussion has been taken from Deza and Deza [4]):

Definition 2. *Given a connected*[4] *graph $G = (V, E)$, its* **path metric** *is a metric on V, defined as the length of a shortest path connecting $x, y \in V$.*

Definition 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***.)*

[2] S_n with composition operation forms the symmetric group of order n.

[3] For $x = (x_i) \in \mathbb{R}^p$, its Euclidean norm is defined as $\|x\| := \left(\sum_{i=1}^{p} x_i^2\right)^{1/2}$

[4] $G = (V, E)$ is connected if for every $x, y \in V$, there exists a path from x to y

Since we are dealing with the decision space of permutations, it would be reasonable to assume that the objective value of candidate solutions is heavily dependent on the *relative orderings* in the permutations. Consider an example in TCP: a test case t_f has high fault detection capability and, consequently, contribute to higher APFD if executed early on. Swapping t_f with the next adjacent test case will delay the fault detection only a little, when compared to swapping t_f with the last test case in the ordering. The distance in relative ordering can be reflected by counting the number of *switches of two adjacent elements* required to move from one permutation to another, formalized as follows:

Definition 4. *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'*.*

Proposition 1. $d_K : S_n \times S_n \to \mathbb{R}_{\geq 0}$ *is indeed a permutation metric.*

The following proposition provides a very intuitive way of computing the swap distance[5]:

Proposition 2. *Given* $\pi, \pi' \in S_n$*,* $d_K(\pi, \pi')$ *is precisely the number of relative inversions between them i.e. number of pairs* $(i, j), 1 \leq i < j \leq n$ *with* $(\pi_i - \pi_j)(\pi'_i - \pi'_j) < 0$*.*

All in all, we want an Euclidean embedding scheme with the following property: embeddings of two permutations are close together if the swap distance between them is small, and vice versa.

Remark 1. One may ask why not stop now and just use the swap distance in SWAY. However, continuous SWAY [2, 3] is designed to be used for Euclidean space, only; specifically, the usage of Euclidean distance is crucial for splitting the candidates via cosine rule, which is not applicable for other metrics.

2.3 Consideration of Naive Embedding

The most trivial Euclidean embedding of permutations would be to take a permutation π of n items directly as a vector in $\mathbb{R}^n$, as mentioned in Definition 1. More formally, it can be defined as follows:

Definition 5. *For fixed n, the* **permutahedron***, denoted as* Π_{n-1}*, is defined as the convex hull of the set* $V_n = \{(\pi(1), \ldots, \pi(n)) \mid \pi \in S_n\}$*, which can be thought of as the "direct" embedding of* S_n *onto* $\mathbb{R}^n$*.*

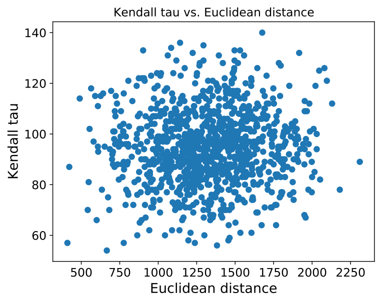
Based on the study of permutahedron in combinatorics [18], we can derive the following propositions about Π_{n-1}.[6]

[5] This proposition indicates a $\mathcal{O}(n \log n)$ algorithm based on sorting. Apart from our consideration, a more efficient algorithm has been proposed; see [1].

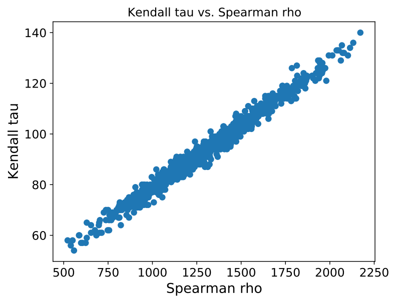
[6] Refer to [24] [19] for the full proofs and more detailed discussions on related topics.

Proposition 3. *Π_{n-1} is a simple $(n-1)$-dimensional polytope with V_n as its set of vertices.*

Proposition 4. *Two vertices of Π_{n-1} are adjacent iff they differ by a swap, when considered as permutations.*



(a) Kendall τ vs. ℓ_2



(b) Kendall τ vs. Spearman ρ

Fig. 1: Scatter plots showing correlations between swap distance and different embedding distances

Simplicity and convexity of the underlying polytope, shown in Proposition 3, ensures that the Split function in Algorithm 1 does not show any unexpected behaviour. What is more interesting is Proposition 4, which *seems* to suggest a positive correlation between the ℓ_2-distance[7] and the swap distance, d_K. However, empirical evidence shows that such correlation either does not exist, or is very weak if it does, rendering the naive and trivial embedding inapplicable. In Figure 1a, the x-axis is the Euclidean distance between two random permutations, $\pi, \pi' \in S^n$, embedded naively using the permutahedron (i.e., $\|\pi - \pi'\|^2$), whereas the y-axis is the swap distance (i.e., $d_K(\pi, \pi')$). No strong positive correlation can be observed. Consequently, we are forced to consider another embedding scheme.

2.4 Motivations for Rank Based Embedding

We start with the following crucial observation: S_n is in bijection with the set of all possible linear orders on $[n]$, denoted as $\mathcal{L}_n$. By dealing with the linear orders instead of permutations, we can leverage several useful results from statistical ranking theory. To start, let us first define such bijection [15]:

Definition 6. *Let $L \in \mathcal{L}_n$.*

[7] For simplicity, let us refer to the distance of the embedded permutations as the ℓ_2-distance of the permutations.

1. *The* **rank function** *of L is the function $r_L : [n] \to [n]$, defined as*

$$r_L(x) = 1 + |\{y \in [n] : \ yLx\}|$$

2. *The* **position permutation** *associated with L is $\pi = (\pi_1 \ \pi_2 \ \cdots \ \pi_n) \in S_n$ with $r_L(\pi_i) = i$ for $i \in [n]$.*
3. *The* **rank permutation** *associated with L is $\pi = (\pi_1 \ \pi_2 \ \cdots \ \pi_n) \in S_n$ with $\pi_i = r_L(i)$ for $i \in [n]$.*

Definition 7. *Let $\pi = (\pi_1 \ \pi_2 \ \cdots \ \pi_n) \in S_n$. Then the linear order $L \in \mathcal{L}_n$ associated with π is defined as $\pi_1 L \pi_2 L \cdots L \pi_n$. Let $r(\pi)$ denote the rank permutation associated with above-defined L, considered as a Euclidean vector.*

We provide a simple example for the concept of rank permutation:

Example 1. Consider $\pi = (2\ 3\ 4\ 1\ 6\ 5) \in S_6$. Then the linear order $<'$ on $[n]$ induced by π' is given as $2 <' 3 <' 4 <' 1 <' 6 <' 5$. Under $<'$, 1 is the 4th ranking element, 2 is the 1st ranking element, and so on. Putting the ranks altogether gives $r(\pi) = (4\ 1\ 2\ 3\ 6\ 5)$.

Above definition motivates another "distance" between two permutations, which can be formally defined as follows:

Definition 8. Spearman ρ distance *of $\pi, \pi' \in S_n$, denoted as $d_S(\pi, \pi')$, is precisely the Euclidean distance between $r(\pi)$ and $r(\pi')$, considering them as vectors (vertices of Π_{n-1} in $\mathbb{R}^n$)*

Proposition 5. *$d_S : S_n \times S_n \to \mathbb{R}_{\geq 0}$ is indeed a permutation metric.*

Our embedding scheme is based on the following non-trivial results by Monjardet [15]. Here, $d_G(\cdot, \cdot)$ is a function from $S_n \times S_n$ to $\mathbb{R}_{\geq 0}$ that is combinatorially well-defined.

Theorem 1 (Monjardet, 1998 [15]).

$$d_S^2(\pi, \pi') = n d_K(\pi, \pi') - d_G(\pi, \pi') \quad \forall \pi, \pi' \in S_n \tag{1}$$

Eq. 1 implies that *if the effect of d_G is insignificant with respect to d_S^2 and* nd_K, then there is a positive correlation between d_S and d_k. To see this, we perform similar experiment as the one with the naive embedding; we sample sufficient number of pairs of permutations, and for each sampled $(\pi, \pi') \in S_n \times S_n$ we plot a scatter plot of $d_S(\pi, \pi')^2$ vs. $d_K(\pi, \pi')$, shown in Figure 1b. Observe how there is an almost linear relationship between the two quantities, which is precisely what we need.

Based on previous discussions, we propose the following Euclidean embedding scheme of S_n (note how r is a bijection from Π_{n-1} to itself):

$$\pi \in S_n \Longleftrightarrow r(\pi) \in \Pi_{n-1} \subset \mathbb{R}^n$$

3 Preliminary Evaluation: Test Case Prioritisation (TCP)

We consider Test Case Prioritisation (TCP) problem [22] as the subject of our proof-of-concept, preliminary evaluation of our embedding scheme. Intuitively, the goal of TCP is to find the optimal ordering of the test cases such that "early fault detection" is maximized, which can be defined as follows [17]:

Definition 9 (Test Case Prioritisation (TCP)). *Given a test suite, T, the set of permutations of T, PT, and a function from PT to $\mathbb{R}$, $f : PT \rightarrow \mathbb{R}$, find $T' \in PT$ such that $(\forall T'')$ $(T'' \in PT)$ $(T'' \neq T')$ $[f(T') \geq f(T'')]$.*

The ideal choice of f would be the function that measures the real fault detection rate of the given ordering. In reality, such measurement is not available before the entire test suite is executed, forcing us to use surrogates such as structural coverage [17,22]. We focus on the coverage-based approach.

3.1 Performance Metrics

We consider two metrics, Average Percentage of Statement Coverage (APSC) and Average Percentage of Fault Detection (APFD) [17], to evaluate the orderings produced by SWAY. Let T be an ordered test suite. APSC measures the rate of coverage achieved, and can be formally defined as follows:

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

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. Note that it is possible to compute APSC using coverage of each test case measured from the previous version of the System Under Test (SUT). APFD, in comparison, measures the actual rate of fault detection *a posteriori*, because information about fault detection is only available after all test cases have been executed. It is defined as follows:

$$APFD(T) = 1 - \frac{TF_1 + \cdots + TF_m}{nm} + \frac{1}{2n} \tag{3}$$

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

3.2 Baseline Approaches

Currently there exists no alternative way of applying SWAY to permutative decision spaces: since ours is the first embedding for such decision spaces, we do not have a direct baseline approaches to compare against. Instead, we simply investigate the feasibility of our embedding by applying it to TCP.

As a basic sanity check of its results, we compare the results from permutative SWAY to those obtained by additional greedy algorithm, which has been widely used in the regression testing literature [8, 14, 23]. Our aim is not to evaluate SWAY itself for TCP problem: a proper evaluation of the efficiency of SWAY would require careful parameter tuning for the sampling size as well as sufficiently optimized implementation. As a proof-of-concept evaluation, we simply check whether our permutative SWAY can produce comparable results to the additional greedy algorithm, and leave the direct comparison between SWAY and other population based optimisation algorithms for future work.

3.3 SWAY for TCP

We make the following changes to the original SWAY to adapt it to the proposed embedding of permutations. Our implementation is available from our GitHub repository.[8]

Initial Sampling For even a medium sized problem, using the entire S_n is infeasible due to the excessive memory required for storing and doing various operations on $n!$ permutations. To avoid this issue, we initialize the population (i.e., samples) by generating random permutations from S_n using Fisher-Yates shuffle[9], which outputs uniformly distributed permutations [6].

Given a set of all test cases, $\mathcal{T} = \{t_1, \ldots, t_n\}$, it can be observed that *each permutation of* $[n]$ *corresponds to a unique ordering of* $\mathcal{T}$, giving us a direct problem-specific interpretation: the decision space of TCP is precisely S_n. The objective score that we are optimizing for is APSC with respect to the already-known execution information of each test case. Typically, this is calculated using the previous version of the SUT.

Stopping Population Stopping population is a user-defined threshold: SWAY stops once the size of the population falls below that threshold. For multi-objective problems, it is natural to set the stopping population to be high [2, 3], since diversity is one of the important quality metrics for such problems [20]. However, if the fitness evaluation is sufficiently expensive for a single objective problem, it would be natural to use SWAY for single objective problems as well.

Suppose that we initially have N candidate solutions as input to SWAY, and set the stopping population as N_0. SWAY will compare all remaining N_0 candidates after stopping. Subsequently, the number of fitness evaluations would be $\mathcal{O}(\log N + N_0)$. Note that if $N_0 = \mathcal{O}(\log N)$, then the number of required fitness evaluations will simply be $\mathcal{O}(\log N)$. We set the stopping population for TCP to be five which is small enough to satisfy such a condition. Out of the five solutions, we choose the one with the highest APSC, and report its APFD.

[8] `https://github.com/chanijung/sway-perm`
[9] This is implemented by `random.shuffle` function in Python

Table 1: Linux utilities

Name	SLOC	$\|T\|$	Description
`flex`	3,453 – 4,008	21	Lexer generator
`gzip`	3,195 – 3,443	193	Data compression utility
`grep`	1.744 – 2,018	211	Pattern matching engine
`sed`	3,729	36	Stream text editor

3.4 Benchmarks

All four Unix utilities we use with our empirical evaluation were obtained from the SIR repository [5]: `flex`, `gzip`, `grep`, and `sed`. Table 1 contains the details of these subject programs and their test suites[10]. The test cases are built to exercise each parameter as well as to achieve coverage, and artificial faults have been injected for the evaluation of regression testing techniques. We consider four versions of `flex`, three versions of `grep`, two versions of `gzip`, and a single version of `sed`.

3.5 Research Questions

We structure our preliminary evaluation of SWAY for permutation decision space around the following research questions:

1. **RQ1. Effectiveness:** How well SWAY performs compared to the greedy prioritisation for TCP?
2. **RQ2. Sensitivity:** How sensitive is SWAY to the initial population?

As mentioned in Section 3.2, we only consider the 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 also consider a random algorithm, which just outputs a random permutation. However, note that the objective of our study is to show the feasibility of our embedding of permutations, and not to present a novel TCP approach.

Each algorithm was executed with a fixed test suite for each program. SWAY and random algorithm were repeated 30 times in order to account for their randomness. For SWAY, the initial population of the candidate permutations was fixed as 2^{19} for all of the programs.[11]

4 Results

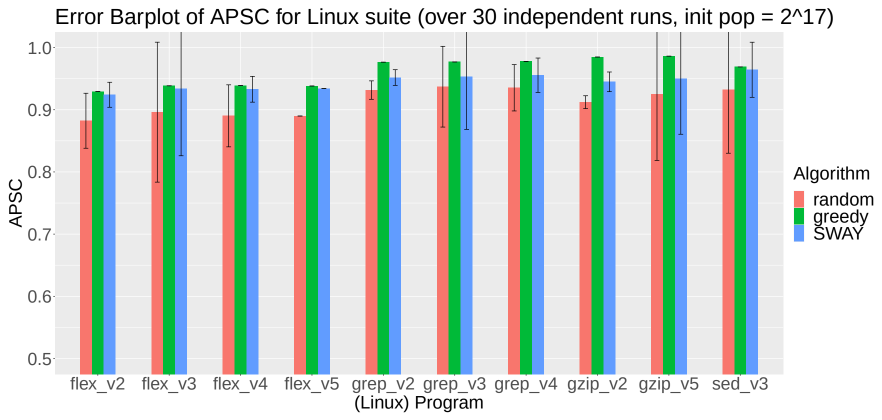
This section presents the results of our preliminary evaluation.

[10] Test suites `v0.cov.universe`, `v0.tsl.universe`, `v0.cov.universe.esc`, and `v0_2.universe.single` have been used for `flex`, `gzip`, `grep`, and `sed`, respectively. Individual test cases that resulted in segmentation fault in our test environment have been filtered out, resulting in smaller test suites than those reported in SIR. Please note that this does not interfere with the feasibility evaluation of our embedding.

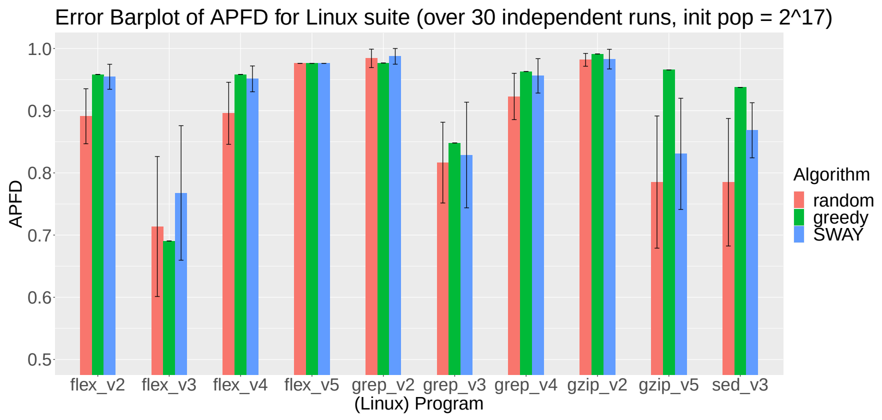
[11] This was the maximum feasible value allowed by our available resource.

4.1 RQ1: How well does our approach perform?

Figures 2a and 2b show the resulting error bar plots of APSC and APFD for the studied subjects. The observable trend is that greedy and SWAY achieve similar APSC values, but SWAY can sometimes outperform greedy, especially for some of the Linux utility runs.



(a) APSC for Linux Util.



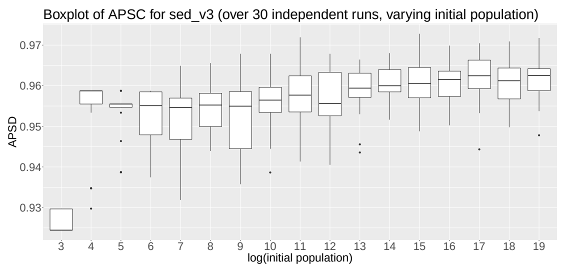
(b) APFD for Linux Util.

Fig. 2: Error bar plots for Linux Utils (RQ1)

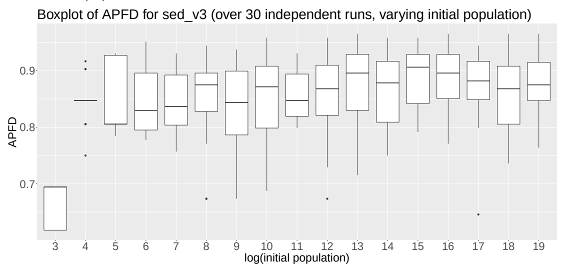
To compare the results of greedy and SWAY more precisely, we apply Mann-Whitney U test [11], a non-parametric test for comparing two statistically independent groups. The null hypothesis, H_0, is that, for X and Y randomly selected from each group, $\Pr[X \geq Y] = \Pr[X \leq Y]$. For each APSC and APFD, we set the alternative hypothesis H_1 as that SWAY performs better than greedy. Table 2 contains the results: statistically significant results are typeset in **bold**.

Table 2: Mann-Whitney U Test of APSC and APFD between SWAY and Greedy

Name	p_{APSC}	p_{APFD}	Name	p_{APSC}	p_{APFD}
`flex-v2`	1.00	0.89	`grep-v3`	1.00	0.97
`flex-v3`	1.00	**0.00**	`grep-v4`	1.00	0.32
`flex-v4`	1.00	**0.02**	`gzip-v2`	1.00	0.97
`flex-v5`	1.00	1.00	`gzip-v5`	1.00	1.00
`grep-v2`	1.00	**0.00**	`sed-v3`	1.00	1.00



(a) APSC vs size of initial population



(b) APFD vs size of initial population

Fig. 3: Boxplots for **RQ2**. Here, log is treated as binary log i.e. $\log_2$.

The results show that greedy outperforms SWAY on APSC in general, which is to be expected since greedy directly (and deterministically) maximizes APSC. However SWAY and greedy algorithm are more at par in terms of APFD: for some programs, SWAY outperforms greedy with statistically significant p-values. Note that, given that the aim of SWAY is to provide an efficient baseline based on random sampling, our aim here is not to consistently outperform greedy. Based on these results, we answer RQ1 that SWAY can produce comparable results to those of the additional greedy algorithm when applied to the TCP problem using the proposed embedding.

4.2 RQ2: Sensitivity of SWAY to Initial Sample Size

We perform the sensitivity analysis only using `sed`, due to the large number of candidate samples required for the sensitivity analysis. We vary the size of the initial population from 2^3 up to 2^{19}, doubling the size at each step. For each size, we run SWAY 30 times to cater for the randomness in sampling.

Figures 3a and 3b show the resulting bar plots of APSC and APFD, respectively. Both APSC and APFD show monotonically increasing trends as the size of the initial population increases: the correlation is stronger with APSC. We posit that this trend is to be expected since, intuitively, greater initial population implies that the search space being covered by SWAY is greater. However, also note that after certain threshold population, increasing it does not seem to have a significant effect on APFD. Consequently, we answer RQ2 that, above certain size, SWAY is not overtly sensitive to the size of the initial population.

While we do not compare SWAY with other population based Evolutionary Algorithms (EAs) in this work, the initial results from `sed` suggest that SWAY can be a viable baseline for TCP against EAs. For example, Epitropakis et al. [8] configures MOEAs with the budget of 25,000 fitness evaluations for SIR Linux utilities, of which `sed` belongs. Figures 3a and 3b indicate that SWAY can achieve stable performance when given a similar number of fitness evaluations.

4.3 Threats to Validity

We depend on widely used coverage profiling tool, `GNU gcov`, to minimise any threat to internal validity in the process of coverage collection. Compared to a Boolean decision space whose size grows exponentially as 2^n, the size of a permutation decision space grows super-exponentially as $n!$, where n is the size of the considered test suite. Using Stirling's approximation[12], it can be easily seen that even with SWAY, we still need $\mathcal{O}(n \log n)$ fitness evaluations when using all possible permutations. Given the size of the studied test suites, it is possible that the observed performance of SWAY has been severely affected by our chosen range of initial populations, posing a threat to external validity. Further studies are needed to explore scalability of SWAY for permutation, as well as other combinatorial decision spaces. Both APSC and APFD are widely studied and used evaluation metrics for the TCP problem, which minimises the threat to construct validity of our study.

5 Related Work

A baseline optimiser is an optimiser that is simple, widely and publicly available, fast, offer comparable performance to the SOTA methods, and computationally inexpensive [21]. It is beneficial to have a baseline optimizer for an optimisation problem because it provides a floor performance values to it. This helps the developers to rule out the optimizers with lower performance. SWAY satisfies all such criteria [2,3], which motivates us to extend it to additional decision spaces.

We compare SWAY to coverage based greedy prioritisation due to the availability of coverage data as well as the ease of implementing the additional greedy

[12] $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$.

algorithm. However, there are numerous TCP techniques that are not focused on early increase of coverage only. For example, history based prioritisation [7, 12] aims to execute the least recently executed test cases first. As a preliminary evaluation, we also focus on single objective TCP for the sake of simplicity. However, Multi Objective Evolutionary Algorithms (MOEAs) have been successfully applied to TCP with multiple objectives [8]. We leave the evaluation of multi-objective formulation of SWAY for TCP for future work.

While the primary intended use of SWAY is a baseline and not an efficient search algorithm in itself, the efficiency of SWAY can be improved by adopting optimised fitness evaluation. For example, SWAY can benefit from the coverage compaction [8] like any other TCP technique, as the compaction can make each fitness evaluation faster.

6 Conclusions and Future Work

This paper presents a novel embedding of the permutations into the Euclidean space, allowing us to directly use (continuous) SWAY without any additional modifications. We base our embedding on well-established fields of combinatorics and statistical ranking theory, and show that our embedding scheme is suitable for the framework of SWAY. A proof-of-concept evaluation of our embedding, applied to TCP using continuous SWAY, shows that it can successfully measure the distance between solutions in permutative decision spaces, enabling SWAY to be applied to such spaces. For future work, we will focus on improving the scalability of SWAY in permutation decision space by applying (non)linear dimensionality reduction techniques after our embedding.

References

1. Chan, T.M., Pătraşcu, M.: Counting inversions, offline orthogonal range counting, and related problems. In: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms. p. 161–173. SODA '10, Society for Industrial and Applied Mathematics, USA (2010)
2. Chen, J., Nair, V., Krishna, R., Menzies, T.: "Sampling" as a Baseline Optimizer for Search-Based Software Engineering. IEEE Transactions on Software Engineering **45**(6), 597–614 (2019)
3. Chen, J., Nair, V., Menzies, T.: Beyond evolutionary algorithms for search-based software engineering. Information and Software Technology **95**, 281–294 (2018)
4. Deza, M.M., Deza, E.: Encyclopedia of Distances. Springer-Verlag Berlin Heidelberg, 4 edn. (2018)
5. Do, H., Elbaum, S., Rothermel, G.: Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. Empirical Software Engineering **10**(4), 405–435 (Oct 2005)
6. Durstenfeld, R.: Algorithm 235: Random permutation. Communications of the ACM **7**(7), 420 (Jul 1964)
7. Engström, E., Runeson, P., Ljung, A.: Improving regression testing transparency and efficiency with history-based prioritization – an industrial case study. In: 2011

Fourth IEEE International Conference on Software Testing, Verification and Validation. pp. 367–376 (2011)

8. Epitropakis, M.G., Yoo, S., Harman, M., Burke, E.K.: Empirical evaluation of pareto efficient multi-objective regression test case prioritisation. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis. pp. 234–245. ISSTA 2015, ACM, New York, NY, USA (2015)
9. Faloutsos, C., Lin, K.I.: Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. SIGMOD Rec. **24**(2), 163–174 (May 1995)
10. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys **45**(1), 11:1–11:61 (December 2012)
11. Hollander, M., Wolfe, D.A., Chicken, E.: Nonparametric Statistical Methods. Wiley Series in Probability and Statistics, Wiley, 3 edn. (2014)
12. Kim, J.M., Porter, A.: A history-based test prioritization technique for regression testing in resource constrained environments. In: Proceedings of the 24th International Conference on Software Engineering. pp. 119–129. ACM Press (May 2002)
13. Lakhotia, K., Harman, M., McMinn, P.: A multi-objective approach to search-based test data generation. In: Proceedings of the 9th Conference on Genetic and evolutionary Computation. pp. 1098–1105 (July 2007)
14. Li, Z., Harman, M., Hierons, R.M.: Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering **33**(4), 225–237 (2007)
15. Monjardet, B.: On the comparison of the spearman and kendall metrics between linear orders. Discrete Mathematics **192**(1), 281 – 292 (1998)
16. Puschner, P., Nossal, R.: Testing the results of static worst–case execution-time analysis. In: 19th IEEE Real-Time Systems Symposium (RTSS '98). pp. 134–143. Los Alamitos, California, USA (1998)
17. Rothermel, G., Untch, R., Chu, C., Harrold, M.: Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering **27**(10), 929–948 (2001)
18. Schoute, P.H.: Analytic treatment of the polytopes regularly derived from the regular polytopes. Verhandelingen der Koninklijke Akademie van Wetenschappen Te Amsterdam **11**(3), 370–381 (1911)
19. Stanley, R.P.: Enumerative Combinatorics: Volume 1, Cambridge Studies in Advanced Mathematics, vol. 49. Cambridge University Press, 2 edn. (2012)
20. Wang, S., Ali, S., Yue, T., Li, Y., Liaaen, M.: A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). pp. 631–642 (2016)
21. Whigham, P.A., Owen, C.A., Macdonell, S.G.: A baseline model for software effort estimation. ACM Transactions on Software Engineering and Methodology **24**(3) (May 2015)
22. Yoo, S., Harman, M.: Regression Testing Minimization, Selection and Prioritization: A Survey. Software Testing Verification and Reliability **22**(2), 67–120 (Mar 2012)
23. Yoo, S., Harman, M.: Pareto efficient multi-objective test case selection. In: Proceedings of International Symposium on Software Testing and Analysis. pp. 140–150. ISSTA 2007, ACM Press (July 2007)
24. Ziegler, G.M.: Lectures on Polytopes, Graduate Texts in Mathematics, vol. 152. Springer-Verlag New York (2007)