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

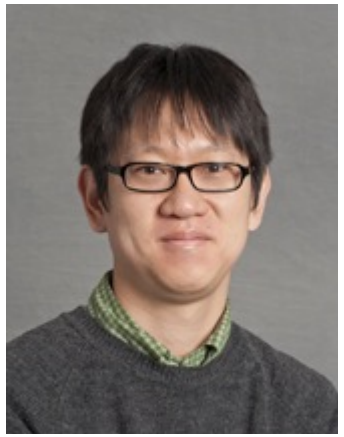
Junghyun Lee*, Chani Jung*, Yoo Hwa Park*, Dongmin Lee*, Juyeon Yoon, Shin Yoo

KAIST

(* equal contribution)





Paper: https://link.springer.com/chapter/10.1007/978-3-030-88106-1_3
Github: https://github.com/chanijung/sway-perm

Introduction

- SBSE often deals with optimisation of **fitness function** over some **decision space**:

$$o = fitness(d)$$

  - $d \in D$: decision variable(space)
  - $\mathrm{o} \in O$: objective variable (space)

- However, population-based metaheuristics often require large number of evaluations.

- Moreover, each fitness evaluation is accompanied by the cost of the actual execution, which may be huge.

Sampling-based Approach

- Key observation:

  There exists a close association between D and O

- How to use above observation to *reduce the number of fitness eval?*

Cluster the candidates by their *decisions* instead of their objectives

- Based on this intuition, Chen et al. [1] proposed SWAY (the **S**ampling **Way**), a FastMAP-type sampling algorithm.

Continuous SWAY

Main routine

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

Sub-routine (Split)

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

Continuous SWAY

Main routine

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

Sub-routine (Split)

Function `Split(`*candidates*`)`**:**
  rand $\longleftarrow$ randomly selected candidate from *candidates*
  east $\longleftarrow \operatorname{argmax}_{x \in candidates} \|x - rand\|$
  west $\longleftarrow \operatorname{argmax}_{x \in candidates} \|x - east\|$
  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

Continuous SWAY

Main routine

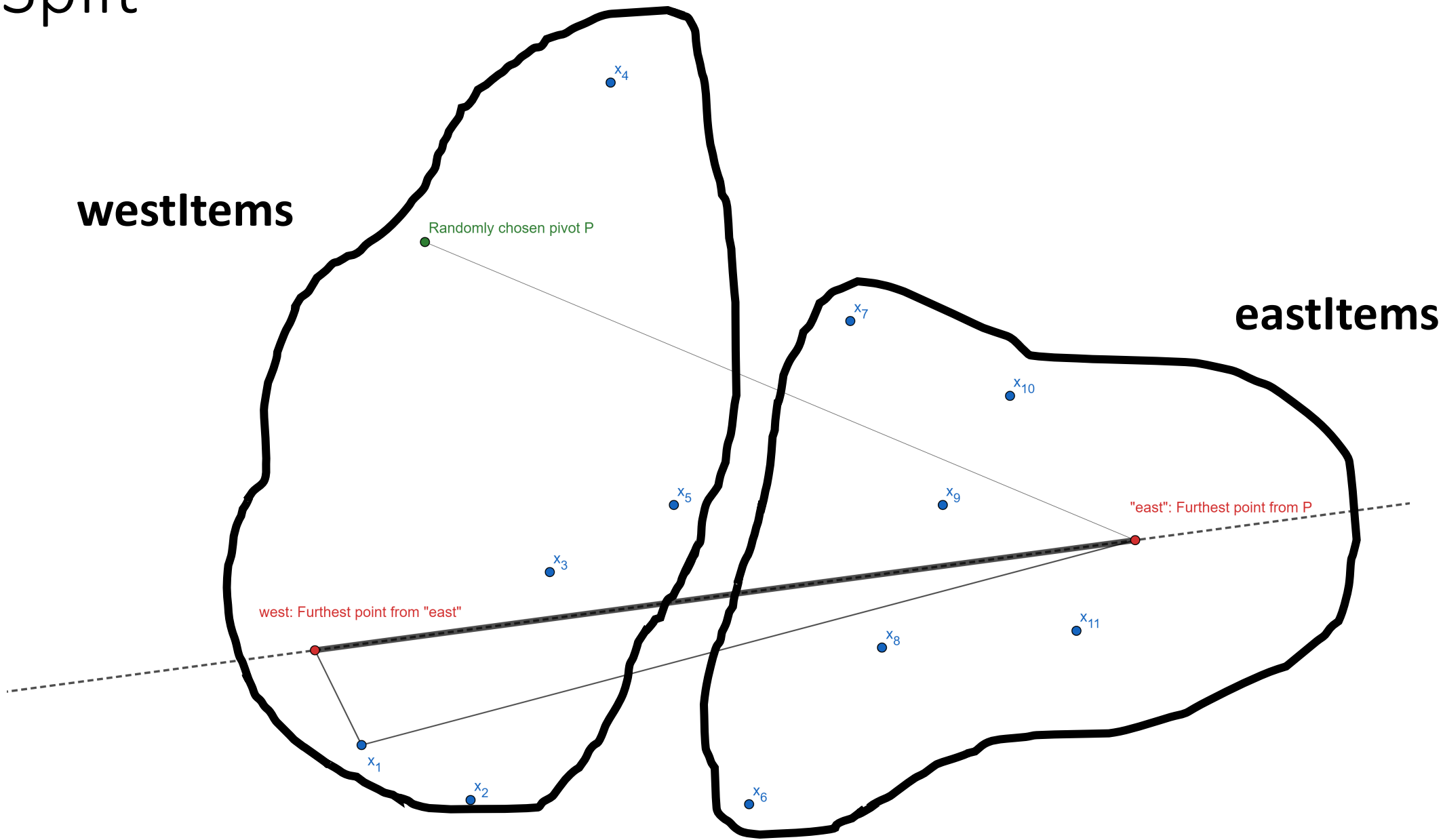
Algorithm 1: Continuous SWAY with its subroutine Split

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

Sub-routine (Split)

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

Split


westItems
eastItems
Randomly chosen pivot P
"east": Furthest point from P
west: Furthest point from "east"
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11

Continuous SWAY

Main routine

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

Sub-routine (Split)

Function `Split(`*candidates*`)`**:**
  rand $\longleftarrow$ randomly selected candidate from *candidates*
  east $\longleftarrow \operatorname{argmax}_{x \in candidates} \|x - rand\|$
  west $\longleftarrow \operatorname{argmax}_{x \in candidates} \|x - east\|$
  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

Some Remarks on SWAY

- Continuous SWAY is heavily dependent on the well-definedness of inner product and distances
  - i.e. ”continuity” of the decision space

18 **Function** `Split(`*candidates*`):`
19 $\quad$ *rand* $\longleftarrow$ randomly selected candidate from *candidates*
20 $\quad$ *east* $\longleftarrow$ $\text{argmax}_{x \in candidates} \|x - rand\|$
21 $\quad$ *west* $\longleftarrow$ $\text{argmax}_{x \in candidates} \|x - east\|$
22 $\quad$ **for** $x \in candidates$ **do**
23 $\quad\quad$ $x_d \longleftarrow \frac{\langle east-west, x-west \rangle}{\|east-west\|}$
24 $\quad$ **end**
25 $\quad$ Sort candidates by x_d
26 $\quad$ *eastItems* $\longleftarrow$ first half of *candidates*
27 $\quad$ *westItems* $\longleftarrow$ second half of *candidates*
28 $\quad$ **return** $[west, westItems], [east, eastItems]$
29 **End Function**

- SWAY was extended to binary case in the same work by Chen et al. (not included here)

Permutation Decision Space

- Many of the SE problems, however, involve *non-binary discrete* decision space.

- One particular case is the **permutation decision space** in which each decision is a permutation!

- Examples: Test Case Prioritisation, Traveling Salesperson Problem

SWAY for Permutation Decision Space?

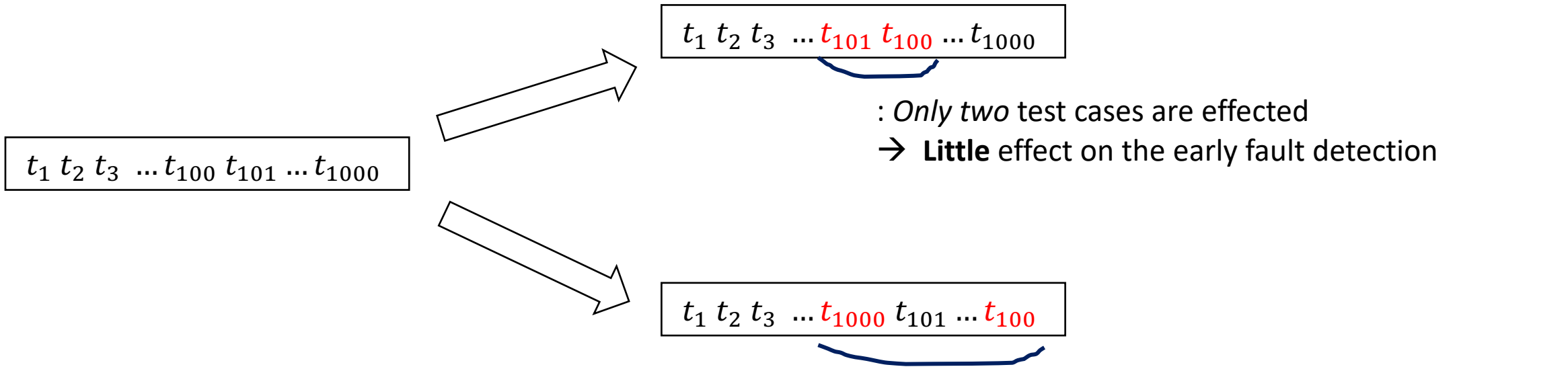
- How to apply SWAY to permutation decision space?

- Coarse-grained binary grouping, then apply binary SWAY? (Chen et al., 2019)
  - It is not clear what coarse-grained groupings can be used in a permutative decision space without knowing which ordering is better than others!

- New embedding of permutations, then apply continuous SWAY? (**this work**)
  - The embedding scheme must preserve some combinatorial distance between permutations

Swap distance between permutations

- Objective value of candidate solutions is heavily dependent on the **relative orderings in the permutation**



Swap distance between permutations

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.*

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* $(\pi_i - \pi_j)(\pi'_i - \pi'_j) < 0$*.*

Find some Euclidean embedding of permutations that best ***preserves the swap distance****!*

Naïve embedding?

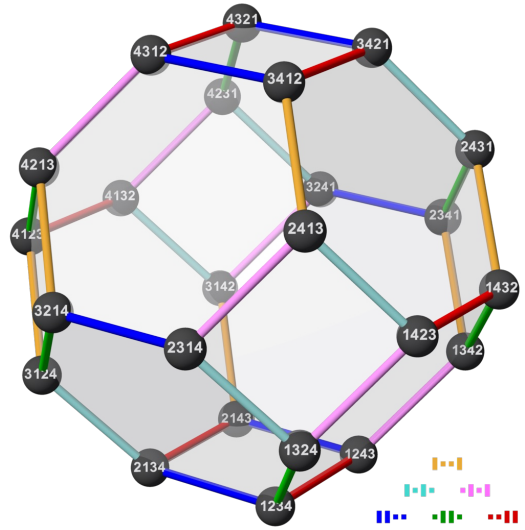
- Consider a very naïve embedding: given some permutation $\pi \in S_n$, its embedding is given as

$$(\pi(1), \pi(2), \ldots, \pi(n)) \in \mathbb{R}^n$$

- If we collect the embedded points of all possible permutations, we get a polytope called *permutahedron*

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 $\mathbb{R}^n$.*

Naïve embedding?



- Permutahedron has many desirable properties:

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

  - : the Split function in Algorithm 1 does not show any unexpected behaviour.

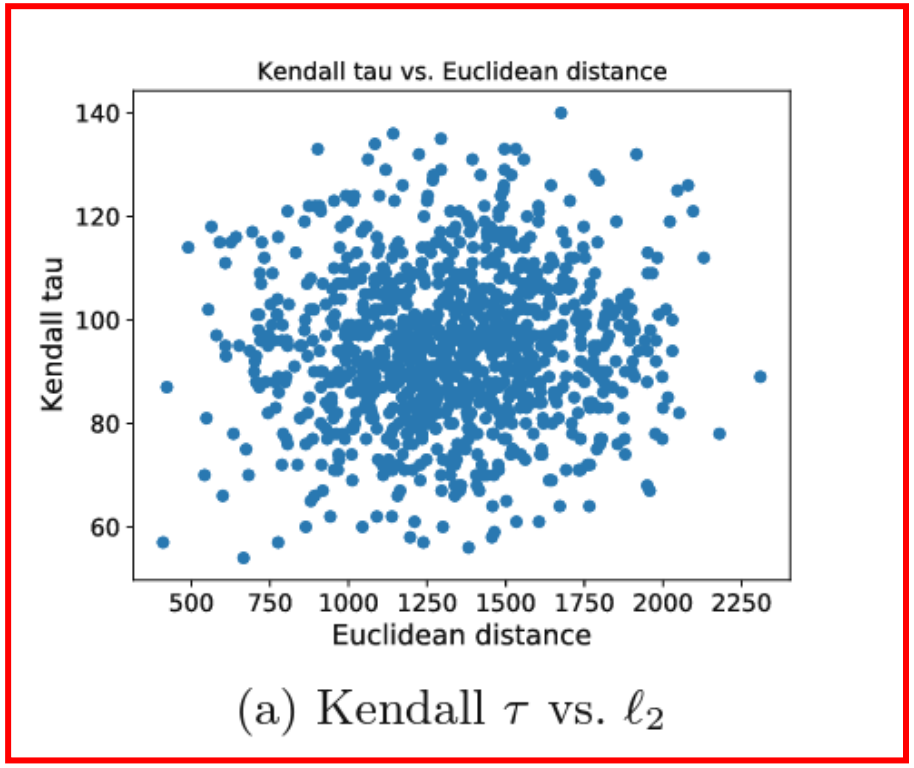
Proposition 4. *Two vertices of Π_{n-1} are adjacent iff they differ by a swap, when considered as permutations.*

  - : *seems* to indicate a positive correlation between the Euclidean distance of embeddings and the swap distance

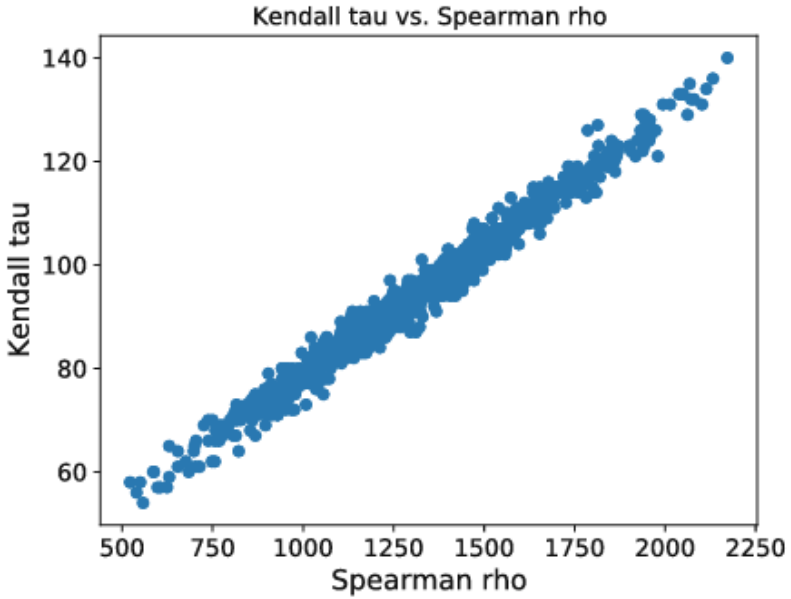
- So why not just use the naïve embedding??

Naïve embedding?

- Turns out, such *seemingly true* property is **false:**



(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

Statistical Ranking 101: rank permutation

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

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.*

Statistical Ranking 101: rank permutation

- Let us go over an example for a clearer understanding!
  - Consider $\pi = (2\ 3\ 4\ 1\ 6\ 5) \in S_n$
  - We can consider a linear order $<'$ induced by π, defined as follows:
    $$2 <' 3 <' 4 <' 1 <' 6 <' 5$$
  - Now, let us consider the ***ranking*** of each number under $<'$
    - 1 is the 4th ranking element, 2 is the 1st ranking element…etc.
  - Putting the ***ranks*** into a single vector gives the rank permutation:
    $$r(\pi) = (4\ 1\ 2\ 3\ 6\ 5) \in S_n$$

Statistical Ranking 101: Spearman rho

- Using rank permutation, another distance between permutations can be considered:

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.*

Rank-based embedding to the rescue!

- Our embedding scheme is based upon the following non-trivial result that *relates* ***Kendall tau*** *with* ***Spearman rho****:*

Theorem 1 (**Monjardet, 1998** [15]).

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

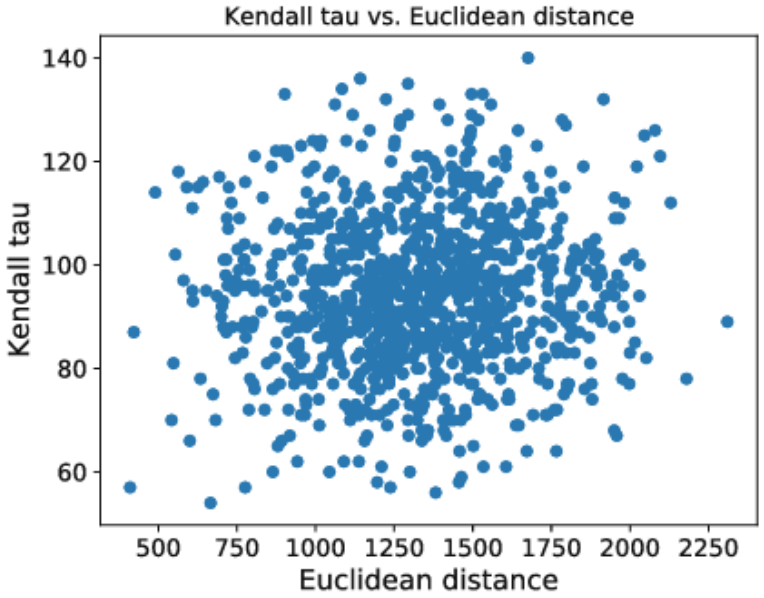
If the effect of d_G *is insignificant with respect to* d_S^2 *and* nd_K, then there is a **positive correlation between** d_S **and** d_K.

i.e. we can embed π as its rank permutation vector:

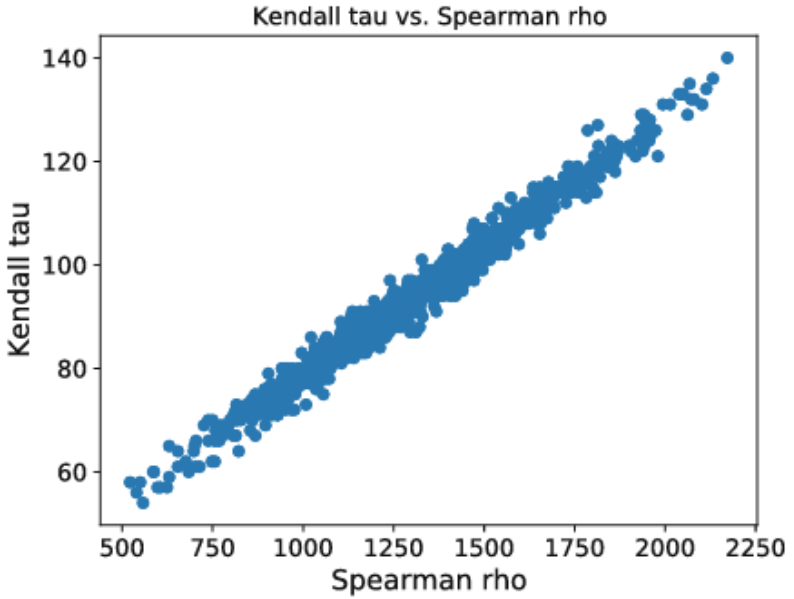
$$\pi \in S_n \Longleftrightarrow r(\pi) \in \Pi_{n-1} \subset \mathbb{R}^n$$

Rank-based embedding to the rescue!

- And indeed, an almost linear relationship can be observed!



(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

Proof-of-concept: TCP

- Let us check the quality of our embeddings by seeing whether applying continuous SWAY gives comparable performance!

- We consider the task of *Test Case Prioritisation (TCP)*
  - Find the optimal ordering of the test cases such that “early fault detection” is maximized

- This is *proof-of-concept,* and thus we do not aim to evaluate SWAY itself for TCP problem!

Performance metrics

- Average Percentage of Statement Coverage (APSC)

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

- Average Percentage of Fault Detection (APFD)

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

Benchmarks

- Four Unix utilities (from SIR repository)
  - **flex** (4 versions)**, gzip** (3 versions)**, grep** (2 versions)**, sed** (1 version)

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

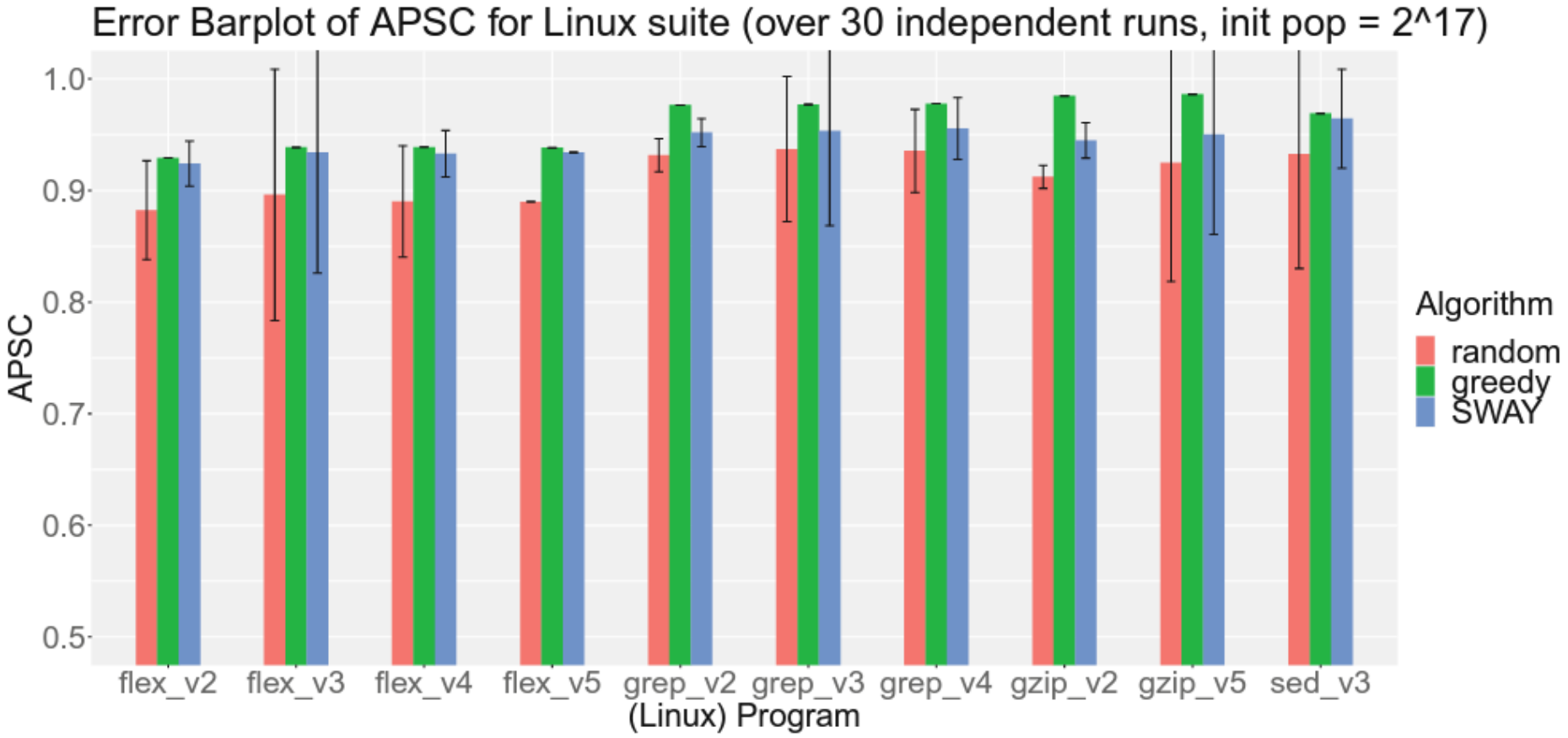
- Individual test cases that resulted in segmentation fault in our test environment have been filtered out, resulting in smaller test suites than those reported in SIR.
- This does *not* interfere with the feasibility evaluation of our embedding.

Algorithms

- Baseline: additional greedy algorithm
  - Widely used in regression testing literature
  - Simple yet efficient

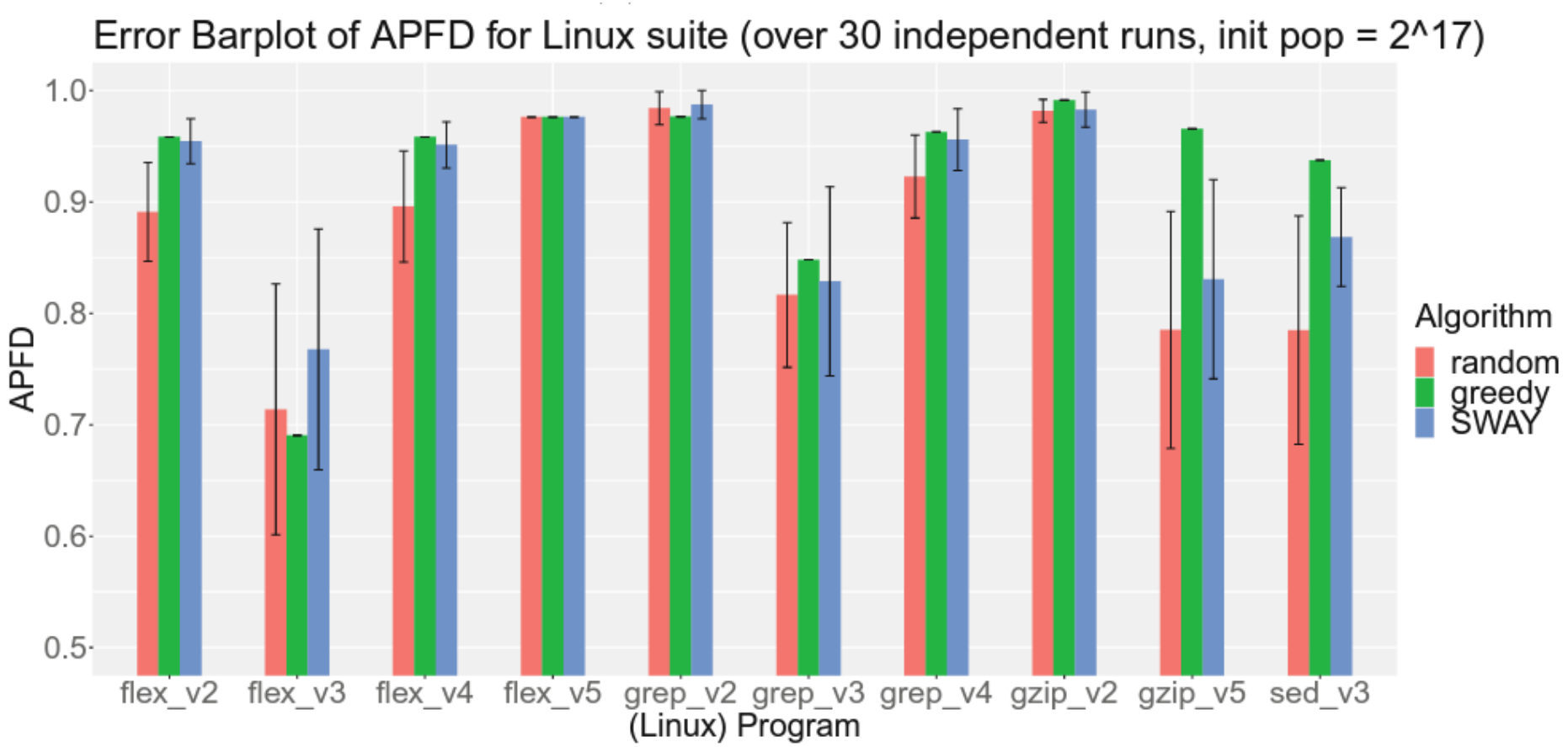
- SWAY for TCP (“Permutative SWAY”)
  - Embed the permutations, then run continuous SWAY
  - **Initial population:** *randomly* sample permutations (fixed as 2^{17})
  - **Stopping population:** sufficiently low value (fixed as 5)

RQ1. Efficacy of Permutative SWAY



(a) APSC for Linux Util.

RQ1. Efficacy of Permutative SWAY



(b) APFD for Linux Util.

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

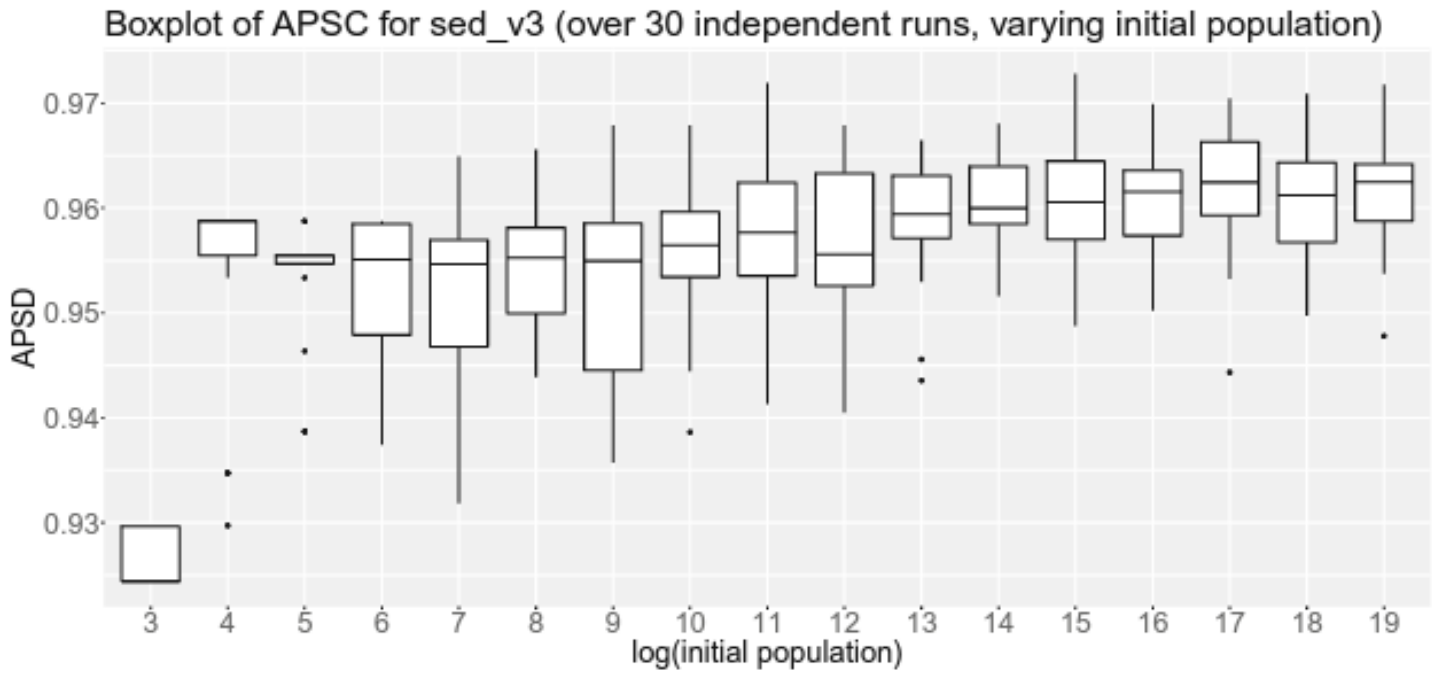
RQ1. Efficacy of Permutative SWAY

- Greedy outperforms SWAY **on APSC** in general
  - This is expected since greedy directly (and deterministically) maximizes APSC.

- SWAY and greedy algorithm are more *at par* in terms **of APFD**
  - For some programs, SWAY outperforms greedy in statistically significant manner.

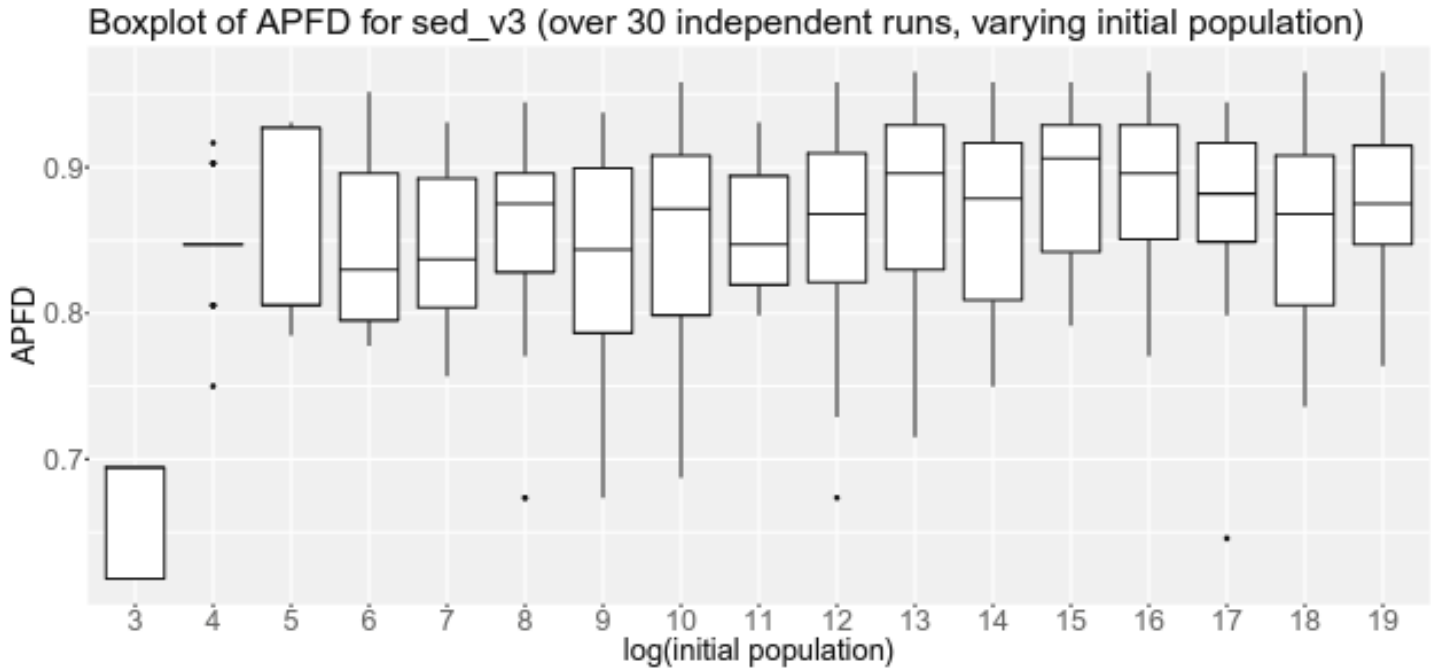
- **RQ1.** SWAY can produce comparable results to those of the additional greedy algorithm when applied to the TCP problem using the proposed embedding.

RQ2. Sensitivity to Initial Population Size



(a) APSC vs size of initial population

RQ2. Sensitivity to Initial Population Size



(b) APFD vs size of initial population

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

RQ2. Sensitivity to Initial Population Size

- Both APSC and APFD show **monotonically increasing trends** as the size of the initial population increases.
  - The correlation is stronger with APSC
- Note that further increasing the initial population after certain threshold does not seem to have a significant effect on APFD.
- **RQ2.** Above certain size, SWAY is not *overtly sensitive to the size of the initial population*.

Contributions

- A novel Euclidean embedding of permutations, allowing for a *direct use of continuous SWAY* for **permutation decision space**

- **Theoretically well-founded** motivation for our embedding scheme
  - Combinatorics, statistical ranking theory

- A proof-of-concept empirical evaluation using TCP

Future works

- Code optimisation and more experiments to further confirm the efficacy.
- Make the current framework more scalable to even huge initial population, as well as the size of the permutation (e.g. number of test cases)
  - ex. Dimensionality reduction, stochastic modification (batching)
- Pseudo-random sampling of the initial permutations to reduce variance?
  - Low-discrepancy sequences (ex. Sobol sequence)
- The clustering used here is very simple.
  - ***Deep representation learning*** to discover underlying complex patterns may result in better quality embeddings. (ex. Autoencoders, deep clustering…)