Implementation and Evaluation of Estimation of Distribution Algorithms in Combinatorial Optimization

Date: 26 May 2020

Author: Gustav Valdemar Fjorder

Advisor: Carsten Witt

Github: github.com/gustavfjorder/Estimation-of-distribution-algorithms

Abstract

Estimation of Distribution Algorithms (EDAs) are a type of optimization algorithm belonging to the field of Evolutionary Computation. Many EDAs have been developed for problems based on the integer or real domain while much fewer have been developed specifically to deal with permutation problems such as the Traveling Salesman Problem (TSP). In this paper, we review a number of EDAs for permutations by testing their abilities to find the identity permutation as well as their ability to find good quality solutions to TSP instances.

Contents

1 Introduction

Estimation of Distribution Algorithms (EDAs) find solutions to problems by building a probabilistic model from which they sample new individuals. New individuals are then factored into the probabilistic model based on some selection criterion so that better solutions are sampled from the model over time. EDAs are said to be an improvement over Genetic Algorithms because EDAs learn a joint probability distribution over the population of promising individuals where relations between different characteristics of the problem can be expressed [1].

In this paper, we document the implementation and evaluation of a selection of EDAs for permutation based optimization problems. First, we present common distance measures for permutations as well as distance measures for presortedness. Then, we present the three EDAs we will evaluate as well as a permutation version of (1+1) EA for benchmarking. Finally, we test and evaluate the distance measures and algorithms using statistical tests.

2 Distance measures

Given permutations σ_1 and σ_2, we wish to quantify the distance between the two permutations. There exist many distance measures for permutations. Kendall Tau, Hamming, and Cayley distance are all known to be effective for distance-based modeling [2].

2.1 Kendall Tau distance

Kendall Tau distance is the minimum number of adjacent transpositions (swaps) needed to obtain one permutation from another [3]. Formally, Kendall Tau distance is given by:

$$d_K(\sigma_1, \sigma_2) = \sum_{l \prec_{\sigma_1} j} 1_{[j \prec_{\sigma_2} l]} \tag{1}$$

where $l \prec_\sigma j$ means that l precedes j in σ.

According to [4], any permutation σ of length n can be uniquely defined by $n-1$ integers representing the distance to the identity permutation $I = (1, 2, ..., n)$ of the positions in the permutation. These integers $V_1(\sigma), V_2(\sigma), ..., V_{n-1}(\sigma)$ are given by:

$$V_j(\sigma) = \sum_{l > j} 1_{[l \prec_\sigma j]} \tag{2}$$

In (2), the identity I is implicit. The general form given by (3) takes any two permutations σ_1 and σ_2. Therefore, we can uniquely define a permutation by $n-1$ integers representing the distance of each position to a reference permutation that does not have to be the identity.

$$V_j(\sigma_1, \sigma_2) = \sum_{l \prec_{\sigma_1} j} 1_{[j \prec_{\sigma_2} l]} \tag{3}$$

Using (1) and (3), we can now express the Kendall's Tau distance as $n-1$ integers as shown in (4) [5]:

$$d_K(\sigma_1, \sigma_2) = \sum_{j=1}^{n-1} V_j(\sigma_1, \sigma_2) \tag{4}$$

This decomposition of Kendall's Tau distance into $n-1$ terms will be used later in this paper.

2.2 Cayley distance

The Cayley distance is the minimum number of transpositions needed to obtain one permutation from another [3]. This differs from the Kendall Tau distance in that the Cayley distance allows transpositions from anywhere to anywhere in the permutation. As such, the distance from the identity to $(4, 2, 3, 1)$ is 1 under the Cayley distance (swap 1 and 4) and 5 under the Kendall Tau distance (swap 4 right three times, then 1 left two times).

2.3 Hamming distance

The Hamming distance counts the number of elements that have incorrect positions in the permutation. As such, given permutations σ_1 and σ_2, the Hamming distance is the number of positions i for which $\sigma_1(i) \neq \sigma_2(i)$ [3]. Formally:

$$d_H = |\{i \,|\, i \in \{1, 2, ..., n\}\,, \sigma_1(i) \neq \sigma_2(i)\}| \tag{5}$$

2.4 Measures of presortedness

In an attempt to analyze the expected running time of evolutionary algorithms, Scharnow, Tinnefeld, and Wegener analyzed different EAs on sorting problems [6]. Measuring the presortedness of a permutation can be done using any of the three distance measures we presented above by using the identity permutation as one of the two permutations between which the distance is found. However, when dealing specifically with the sorting problem, Scharnow et al. present two additional distance measures.

2.4.1 Number of sorted blocks ($RUN(\sigma)$)

Given a permutation σ, this is the number plus one of indices i such that $\sigma(i) > \sigma(i+1)$. As such, it measures the number of adjacent pairs that are ordered incorrectly.

2.4.2 Length of longest ascending subsequence ($LAS(\sigma)$)

The length of the longest ascending subsequence is an indication of how far we have come towards sorting the permutation. $LAS(\sigma)$ is 1 for a permutation ordered from largest to smallest and n for the identity. Formally, given a permutation σ, LAS is the largest k such that some $\sigma_i < \sigma_{i+1} < ... < \sigma_{i+k}$.

Since we want the distance to decrease as we approach the identity, we use the negative of this measure. As such, when an algorithm minimizes the $-LAS$ value of a permutation, it maximizes the length of the longest ascending subsequence.

3 Algorithms

This section will present the three EDAs that we want to evaluate and an EA for benchmarking. First, we present the Univariate Marginal Distribution Algorithm which has been adapted to deal with permutations. Second, we present the Edge Histogram Based Sampling Algorithm. Third, we present the Generalized Mallows Model. Lastly, we present the well-known (1+1)EA which has been adapted to deal with permutations.

3.1 Univariate Marginal Distribution Algorithm (UMDA)

UMDA is a simple EDA over bitvectors that samples N individuals and keeps the M best [7]. The frequency p_i is then set equal to the fraction of the M individuals that have 1 at position i. We sample a new individual $x = (x_1, x_2, ..., x_n)$ using $P[x_i = 1] = p_i$ such that the probability of putting a 1 on position i is equal to the fraction of the M individuals in the previous generation that had a 1 at position i.

Here, we propose a variant UMDA$_p$ of UMDA for permutations. Instead of just one value p_i for each position in the permutation, UMDA$_p$ has a vector $\mathbf{p}_i$ of probabilities such that $\mathbf{p}_{ij}$ is the probability that position i gets value j. Therefore, the matrix P has size $n \times n$ and has $\frac{1}{n}$ at every spot initially. When a $x_i = j$ has been sampled, we sample x_{i+1} such that we cannot choose j again to ensure that we get a valid permutation. This is done by maintaining a copy of P where we set $P_{ij} = 0\ \forall i$ and normalize each p_i so the probabilities sum to 1.

With a population size of N, we sample $2N$ individuals and keep the N best ones. Now, we update P as shown in (6) such that P_{ij} is the number of times j appears at position i in the selected population. We add an ε to avoid getting 0 probabilities. Finally, we normalize such that the probabilities in p_i sum to 1 by dividing by $\varepsilon n + N$.

$$P_{ij} = \frac{\varepsilon + |\{x \in \mathit{population} \mid x_i = j\}|}{\varepsilon n + N} \tag{6}$$

As an example, consider running UMDA$_p$ with permutation size $n = 4$, population size $N = 3$, and $\varepsilon = 0.1$. We first initialize P to $\frac{1}{n}$:

$$P = \begin{bmatrix} 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{bmatrix}$$

We now sample $2N = 6$ new individuals and select the $N = 3$ best ones:

$$\begin{aligned} \sigma_1 &= (1, 2, 4, 3) \\ \sigma_2 &= (2, 3, 4, 1) \\ \sigma_3 &= (1, 4, 2, 3) \end{aligned} \tag{7}$$

And we then update P. Looking at position 1 of the three individuals, there are two 1s and one 2. Thus, row p_1 will be given by:

$$\begin{aligned} p_1 &= \left(\frac{0.1 + \mathbf{2}}{0.1 \cdot 4 + 3}, \frac{0.1 + \mathbf{1}}{0.1 \cdot 4 + 3}, \frac{0.1}{0.1 \cdot 4 + 3}, \frac{0.1}{0.1 \cdot 4 + 3} \right) \\ &= (0.62, 0.32, 0.03, 0.03) \end{aligned} \tag{8}$$

Therefore, when we sample in the next generation, the value at position 1 will be selected with the probabilities: $P[x_1 = 1] = 0.62$, $P[x_1 = 2] = 0.32$, and $P[x_1 = 3] = P[x_1 = 4] = 0.03$.

3.2 Edge Histogram Based Sampling Algorithm

The Edge Histogram Based Sampling Algorithm (EHBSA) is an EDA described by Tsutsui et al in [8]. The authors view the sampling of a permutation as the creation of a tour through a set of nodes in a graph where inserting an edge from node 1 to 2 corresponds to placing element 2 after 1 in the permutation.

The EHBSA generates an initial population of solutions at random. A selection of the best solutions is made using some selection criterion. Then, an Edge Histogram Matrix (EHM) is created for the selected individuals as shown in (9).

$$e_{i,j} = \begin{cases} \sum_{k=1}^{N} (\delta_{i,j}(\sigma_k) + \delta_{i,j}(\sigma_k)) + \varepsilon & \text{if } i \neq j \\ 0 & i = j \end{cases} \tag{9}$$

Where the δ-function given by (10) returns 1 if i directly precedes j. Notice that we check for $(h+1) \bmod n$ which means we wrap around the end of the permutation such that in $(1,2,3,4,5)$ we say that 5 directly precedes 1.

$$\delta_{i,j}(\sigma_k) = \begin{cases} 1 & \text{if } \exists h[h \in \{0,1,...,n-1\} \wedge \sigma_k(h) = i \wedge \sigma_k((h+1) \bmod n) = j] \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The ε creates a bias toward random permutations. In the article they suggest using $\varepsilon = \frac{2N}{n-1} B_{ratio}$ to use comparable pressure for all problems. The $B_{ratio} > 0$ is suggested set to 0.04 which we do in the implementation.

Two sampling algorithms are suggested: a simple one and a more complex one that uses templates. The two will be called EHBSA/WO and EHBSA/WT, respectively.

EHBSA/WO starts by sampling a random element which gets position 1 in the permutation. The following $n-1$ elements are sampled such that position i gets value j with probability $e_{i-1,j}$. The probability of sampling j is then set to 0 for the remaining elements to avoid sampling the same element twice.

EHBSA/WT attempts to improve the sampling by selecting a template individual from the population. A new individual is sampled by taking part of the template individual and generating the remaining positions using the EHM. The elements that are part of the template are decided using a method inspired by n-point crossover. $n > 1$ positions are sampled at random such that the template individual is divided into n segments. One of these segments is chosen randomly and used as the template. The rest of the elements are sampled using the EHM.

Tsutsui notes that EHBSA only works well on problems where the absolute position of an element does not matter such as TSP. In TSP, $(1,2,3,4)$ and $(2,3,4,1)$ have the same tour lengths but in sorting, one is much better than the other. EHBSA learns which elements should be adjacent, but samples an individual by selecting a random starting element for position 1. Thus, even with a perfect probability matrix where we sample i after $i-1$ with probability 1, we will only select 1 as the starting element in $\frac{1}{n}$ samples. Therefore, we predict that EHBSA will not do well on the sorting problem. EHBSA/WT is predicted to do better, but it is unclear how much the template cuts will help.

3.3 EDAs using Mallows Models

In the following, we will present the Mallows Model and the Generalized Mallows Model. They are very similar so we will simply present the Generalized Mallows Model and then point out its differences from the Mallows Model afterwards.

3.3.1 The Generalized Mallows Model

The Generalized Mallows Model is a family of distance-based exponential probability models over permutations [5][9]. Using some distance measure D over permutation spaces, the Generalized Mallows Model is given by:

$$P(\sigma) = \frac{e^{-\theta D(\sigma,\sigma_0)}}{\psi(\theta)} \tag{11}$$

where θ is the spread parameter, σ_0 is the central permutation, and $\psi(\theta)$ is a normalization constant. As such, the probability of sampling a permutation σ depends only on θ and σ_0. The normalization constant $\psi(\theta)$ does not depend on σ_0 and is defined by:

$$\psi(\theta) = \prod_{j=1}^{n-1} \psi_j(\theta_j) = \prod_{j=1}^{n-1} \frac{1 - e^{-(n-j+1)\theta_j}}{1 - e^{-\theta_j}} \tag{12}$$

As (12) shows, $\psi(\theta)$ is the product of $n-1$ terms $\psi_j(\theta_j)$. Thus, the model given by (11) can be factorized into $n-1$ independent univariate exponential models [4]. Thus, instead of defining a probability over permutations, we will use V_j defined in (3) to define the probability of $V_j(\sigma,\sigma_0)$ taking on certain values:

$$P[V_j(\sigma,\sigma_0) = r] = \frac{e^{-\theta_j r}}{\psi_j(\theta_j)} \tag{13}$$

Therefore, the probability of sampling a permutation will now be given as a vector of $n-1$ values representing the relative distances between elements of the sampled permutation and the central permutation σ_0.

3.3.2 Learning a Generalized Mallows Model

A Mallows Model is defined by a central permutation σ_0 and a spread parameter θ which for the Generalized Mallows Model is a vector of θ_j. Assume we have a population of permutations from which we wish to learn a Mallows Model. Finding σ_0 is called rank aggregation and is NP-hard [10]. Therefore, we estimate σ_0 using the Borda algorithm by taking the average value at each position and assigning 1 to the position with the lowest average value, 2 to the second, and so on [9][11].

For example, consider a population of three permutations for which we wish to estimate the central permutation σ_0:

$$\begin{aligned} \sigma_1 &= (5,1,2,3,4) \\ \sigma_2 &= (1,5,4,3,2) \\ \sigma_3 &= (1,3,2,5,4) \end{aligned} \tag{14}$$

We find the average value at each position of the permutation:

$$\sigma_{avg} = \frac{1}{3}(7,9,8,11,10) \tag{15}$$

And then we assign 1 to the position with the smallest average, 2 to the second, and so on:

$$\sigma_0 = (1, 3, 2, 5, 4) \tag{16}$$

Which gives us an estimate for the central permutation. With a population of N permutations of size n, this central permutation estimation takes time $O(n\,N)$ [9].

The spread parameter θ can now we computed using the central permutation σ_0. Each θ_j is calculated by solving the equation [9]:

$$\bar{V}_j = \frac{1}{e^{\theta_j} - 1} - \frac{n - j + 1}{e^{\theta_j\,(n-j+1)} - 1} \tag{17}$$

where $\bar{V}_j$ is the observed mean for V_j, i.e. $\bar{V}_j = \frac{1}{N}\sum_{j=1}^{N} V_j(\sigma, \sigma_0)$. Equation (17) is solved using *scipy.optimize.newton* which is a numerical solver using the Newton-Raphson method [12]. We use a tolerance of 0.001 and maximum iterations of 500. In the cases where the numerical solver does not converge, we set $\theta = 0.1$. This happens when we have a population of size n or less.

3.3.3 Sampling a Generalized Mallows Model

We have learned a Generalized Mallows Model and wish to sample a new permutation σ_s. We use the probability distribution given by Equation (13) to sample a vector $V(\sigma_s\sigma_0^{-1})$ of $n - 1$ integers which we will call $V(\pi)$. This vector uniquely defines some permutation with the central permutation as reference. Each value in $V(\pi)$ defines how many positions an element will move, so the sum of the values in the vector is equivalent $d_K(\sigma_s, \sigma_0)$.

In order to get the new permutation, we apply $V(\pi)$ to the central permutation. To do so, we first find π^{-1} using the algorithm given by [5] and as pseudocode by [9]. We convert π to π^{-1} by inserting the largest element n into an empty list. Then, each element k in descending order $n-1, ..., 1$ is inserted at position $V_k(\pi)$. For example, consider the conversion of $V(\pi)$ to π^{-1} when $V(\pi) = (2, 0, 2, 1)$:

$$\begin{aligned}
&\text{insert } n = 5 \text{ in } \qquad 0 \to \pi^{-1} = (\mathbf{5}) \\
&\text{insert } j = 4 \text{ in } V_4 = 1 \to \pi^{-1} = (5, \mathbf{4}) \\
&\text{insert } j = 3 \text{ in } V_3 = 2 \to \pi^{-1} = (5, 4, \mathbf{3}) \\
&\text{insert } j = 2 \text{ in } V_2 = 0 \to \pi^{-1} = (\mathbf{2}, 5, 4, 3) \\
&\text{insert } j = 1 \text{ in } V_1 = 2 \to \pi^{-1} = (2, 5, \mathbf{1}, 4, 3)
\end{aligned} \tag{18}$$

We now have $\pi^{-1} = (\sigma_s\sigma_0^{-1})^{-1}$ which we invert to get $\pi = \sigma_s\sigma_0^{-1}$. The inversion simply sets at position j the index of the element with value j. This is shown for the resulting $\pi^{-1} = (2, 5, 1, 4, 3)$ from (18):

$$\begin{aligned}
(\pi^{-1})^{-1} &= (2, 5, 1, 4, 3)^{-1} \\
&= (3, 1, 5, 4, 2)
\end{aligned} \tag{19}$$

We now have $\sigma_s\sigma_0^{-1}$ which can be composed with the central permutation to get the new sampled permutation $\sigma_s\sigma_0^{-1}\sigma_0 = \sigma_s$. Using the central permutation from (16) we get:

$$\begin{aligned}
(\sigma_s\sigma_0^{-1})\sigma_0 &= (3, 1, 5, 4, 2)(1, 3, 2, 5, 4) \\
\sigma_s &= (3, 5, 1, 2, 4)
\end{aligned} \tag{20}$$

To verify the sampling, consider $V(\pi) = (2, 0, 2, 1)$ with a sum of 5. The new σ_s has five pairwise disagreements with σ_0 which means we have found a permutation with a Kendall distance of 5.

3.3.4 The Mallows Model

A simplification of the Generalized Mallows Model is the Mallows Model [10]. Where the Generalized Mallows Model has $n-1$ spread parameters θ_j, the Mallows Model uses just one. This means that in the sampling step, the values given to each V_j are sampled using the same spread parameter. We will therefore see each position in the permutation get moved the same distance on average, whereas the Generalized Mallows Model gave different spread parameters to each position in the permutation.

This changes the learning and sampling steps slightly. When learning a Mallows Model, we no longer find $n-1$ values for θ_j by solving Equation (17), instead we find just one θ by solving Equation (21).

$$\sum_{j=1}^{n-1} \bar{V}_j = \frac{n-1}{e^{\theta}-1} - \sum_{j=1}^{n-1} \frac{n-j+1}{e^{(n-j+1)\theta}-1} \tag{21}$$

The sampling step of the Generalized Mallows Model used θ_j as shown in Equation (13). Now, we just use the θ we found for sampling every V_j. This means that each position in the permutation will on average be moved the same distance. We predict that this will lead to longer optimization times since correctly placed elements will be moved at the same frequency as incorrectly placed elements.

3.4 (1+1)EA for permutations

The well-known (1+1) EA optimizes a bitstring by flipping each bit with probability $\frac{1}{n}$ in each iteration where n is the length of the bitstring. The expected optimization time on OneMax is $O(n\,log\,n)$ [13]. Here, we wish to develop an analogous algorithm for permutations.

(1+1) EA for permutations

(1+1)EA for permutations creates a random initial permutation. Then, in each iteration, every element is swapped with a randomly selected element with probability $\frac{1}{n}$ where n is the size of the permutation. If the resulting permutation is better than the previous one, we save it. We then go to the next iteration. Pseudocode for the algorithm is shown in Algorithm 1.

Analysis

We will now analyse the expected optimization time of (1+1) EA for permutations using the method given by Carsten Witt in the course *Computationally Hard Problems*. We will do so by analyzing the time it takes to find the identity permutation. Let ϕ denote the number of elements in σ that are in the right position corresponding to the Hamming distance. We divide the algorithm into phases. Phase i starts when $\phi = i$ and ends when it increases. To get an increase in ϕ, it is sufficient to swap an element at an incorrect position with the element at its correct position. The probability of ϕ increasing in a given run corresponds to the probability of choosing an element that is not in the correct position, swapping it with the element at the correct position, and not moving anything else. We have $n-i$ incorrectly placed elements that are chosen with probability $\frac{1}{n}$ and swapped with its correct position with probability $\frac{1}{n}$. We do not move anything else with

Algorithm 1 (1+1)EA for permutations

```
1: function $(1+1)EA\_Permutations()$
2:     $t \leftarrow 0$                                          ▷ Start time
3:     $\sigma \leftarrow P(\{1,...,n\})$                        ▷ Initial permutation
4:     while $t < t_{max}$ do
5:         $\sigma' \leftarrow \sigma$                           ▷ Make copy of $\sigma$
6:         for $i \in \{1,...,n\}$ do
7:             w.p. $\frac{1}{n}$ swap $\sigma'_i$ with a randomly chosen $\sigma'_j$    ▷ Swap two elements
8:         end for
9:         if $f(\sigma') \geq f(\sigma)$ then                   ▷ If new solution is better, keep it
10:            $\sigma \leftarrow \sigma'$
11:        end if
12:    end while
13:    Return $\sigma$
14: end function
```

probability $(1-\frac{1}{n})^{n-1}$. Therefore, the probability of increasing ϕ given that $\phi = i$ is:

$$\begin{aligned} Pr[increase\ \phi \,|\, \phi = i] &= (n-1) \cdot \frac{1}{n} \cdot \frac{1}{n} \cdot (1-\frac{1}{n})^{n-1} \\ &\geq (n-i) \cdot \frac{1}{n^2} \cdot \frac{1}{e} \\ &= \frac{n-i}{en^2} \end{aligned} \tag{22}$$

Therefore, the expected length of phase i is:

$$E[\text{length of phase } i] = \frac{1}{\frac{n-i}{en^2}} = \frac{en^2}{n-i} \tag{23}$$

So the expected duration of all phases is:

$$\begin{aligned} E[\text{duration of all phases}] &= \sum_{i=0}^{n-1} \frac{en^2}{n-i} \\ &= en^2 \sum_{i=0}^{n-1} \frac{1}{n-i} \\ &= en^2 \sum_{i=1}^{n} \frac{1}{i} \\ &= O(n^2 \log n) \end{aligned} \tag{24}$$

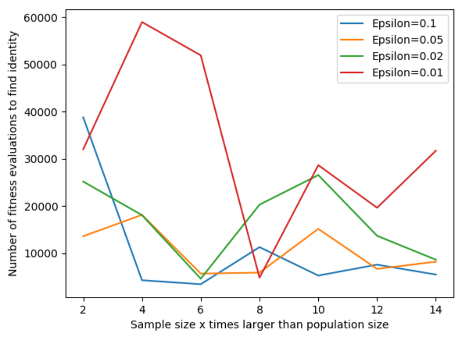

Figure 1: Test of UMDA$_p$ on sorting problem of size 15

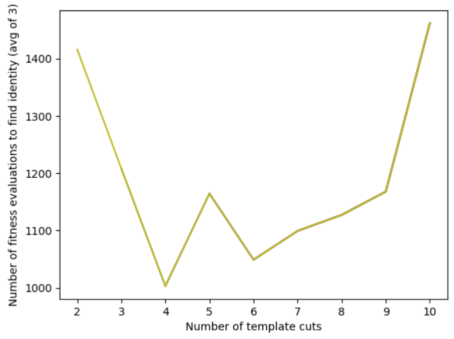

Figure 2: EHBSA/WT performance with varying number of template cuts

4 Testing

4.1 Parameter tuning

4.1.1 Sample size and ε for UMDA$_p$

We chose to use a population size N equal to the size n of the permutation. The sample size was initially set to $2N$ as described in Section 3.1. However, to see if we could quickly find a better choice of sample size, we ran the algorithm on the sorting problem of size 15 with sample sizes in the range $2N$ to $20N$. In the same test, we wanted to find a value for ε that consistently gives good results.

The result of this test is shown in Figure 1. Based on the result, we will use $\varepsilon = 0.05$ and set the sample size to $8N$.

4.1.2 EHBSA with template

The number of template cuts in EHBSA/WT is important for the performance of the algorithm. Figure 2 shows the performance of EHBSA/WT with different numbers of template cuts. Based on the test, we will use 4 template cuts.

4.2 Statistical methods

4.2.1 Friedman's test

We wish to statistically test whether some distance measures are better than others, and whether some algorithms are better than others. To do so, we will use Friedman's test [14]. Friedman's test is a two-way analysis of ranks where the null hypothesis that we wish to reject states that the treatments are sampled from the same population, which in our case corresponds to stating that there is not a statistical difference between the performance of the distance measures or algorithms. With K treatments and n blocks, the test statistic is given by:

$$T = \frac{12}{nK(K+1)} \sum_{k=1}^{K} R_k^2 - 3n(K+1) \tag{25}$$

where R_k is the sum of ranks for a treatment. The null hypothesis is rejected if $T \geq \chi^2_{K-1,1-\alpha}$ which is the $1-\alpha$ quantile of the Chi-squared distribution using $K-1$ degrees of freedom.

4.2.2 Pairwise Comparisons

Rejecting the hypothesis that there is no statistical difference between the treatments/algorithms will not always be so interesting. For instance, if we can easily see that some algorithms are bad then this conclusion will not tell us much. In reality, we might want to know if the best algorithm is statistically superior to the second best algorithm. To test this, we will compare all pairs of treatments [15]. To compare the ith and jth algorithms, we use the test statistic given by:

$$z = \frac{R_i - R_j}{\sqrt{\frac{k(k+1)}{6N}}} \tag{26}$$

Where R_i is the sum of ranks for treatment i. We use the z value to find the corresponding p value from the normal distribution which can then be compared to our $\alpha = 0.05$.

4.3 Performance comparison under different distance measures

To evaluate the distance measures given in Section 2, we will run UMDA$_p$, EHBSA/WT, and the Generalized Mallows Model on the sorting problem of size 10 using the three standard distance measures as well as the two additional measures for presortedness with a time limit of 120 seconds. Population sizes are $\lceil\sqrt{n}\rceil = \lceil\sqrt{10}\rceil = 4$ for UMDA$_p$ and EHBSA/WT, and $10n = 10 \cdot 10 = 100$ for the Generalized Mallows Model. The number of fitness evaluations needed by each algorithm to find the identity under each distance measure is given in Table 1 as an average of three runs where a blank cell means that the identity was not found for any of the three runs. We rank the distance measures such that for each algorithm, the best distance measure gets rank 1 and the worst gets rank 5 as shown in Table 2.

We will now calculate the Friedman statistic using $K = 5$ algorithms and $n = 3$ blocks. The test

Algorithm	**Distance Measure**				
	Hamming	**Kendall**	**Cayley**	**RUN**	**LAS**
UMDA	299	2624	1269	19680	76448
EHBSA/WT	44938	247	79730		35888
GeneralizedMallowsModel	4126	463	1552		

Table 1: Fitness evaluations needed to find identity under different distance measures

Algorithm	**Distance Measure**				
	Hamming	**Kendall**	**Cayley**	**RUN**	**LAS**
UMDA	1	3	2	4	5
EHBSA/WT	3	1	4	5	2
GeneralizedMallowsModel	3	1	2	4.5	4.5
Sum of ranks	**7**	**5**	**8**	**13.5**	**11.5**

Table 2: Distance measure ranked by ability to find identity

statistic is given by:

$$\begin{aligned} T &= \frac{12}{nK(K+1)} \sum_{k=1}^{K} R_k^2 - 3n(K+1) \\ &= \frac{12}{3 \cdot 5(5+1)}(49 + 25 + 64 + 182.25 + 132.25) - 3 \cdot 3(5+1) \\ &= 6.3 \end{aligned} \tag{27}$$

We find the χ^2 value for $df = K - 1 = 4$ and $\alpha = 0.05$ which is 9.488. Since 6.3 is not greater than this value, we cannot reject the null hypothesis.

It is interesting that the Hamming distance is so much better than the other algorithms when running UMDA. This is because the Hamming distance and the probabilistic model of UMDA go well together. When selecting the best individuals, the Hamming distance will prefer those with more elements in the correct positions. This means we will be more likely to sample elements in the correct positions in the next iteration. For EHBSA/WT and the Generalized Mallows Model, the Kendall distance appears superior. This corresponds to the overall consensus in the literature saying that the Kendall distance is the all-round best distance measure over permutations [9].

4.4 Finding identity permutation

We will evaluate the EDAs by testing their abilities to find the identity permutation. We will evaluate their performance based on the number of fitness evaluations used. The CPU time will also be recorded but since we have given the algorithms varying levels of focus in terms of implementation efficiency, this measure will be very biased. We will use population sizes of $\lceil \sqrt{n} \rceil$, n, and $10n$ on permutations of lengths 5, 10, 15, 20, and 25. The results are shown in Tables 3, 4, and 5 where a cell contains the average number of fitness evaluations needed for the given algorithm to find the identity, followed by the number of successful runs out of five, i.e. `70 (5)` means the average number of fitness evaluations is 70, and the algorithm found the identity within the time limit in all five runs. Empty cells indicate that the algorithm was unable to find the identity in any of the five runs.

It is clear that EHBSA/WT is best when the population size is $\sqrt{n}$ and n, and the Generalized Mallows Model is best when the population size is $10n$. To get a fair comparison, we will for each algorithm use the population size for which the algorithm achieved the best average. This result is shown in Table 6. We then rank the algorithms such that for a given problem instance, the algorithm which achieved the best average gets rank 1, and so on. This is shown in Table 7. In the cases where an algorithm was unable to find a solution, the algorithm gets the worst rank and in case of ties, we assign the average of the positions in question such that the sum of ranks remains the same.

We will now see if there is a statistical difference between the algorithms using Friedman's test [14]. The null hypothesis states that there is not a statistical different between the performances of the algorithms.

We will now calculate the Friedman statistic using $K = 6$ algorithms and $n = 5$ blocks. The test statistic is given by:

$$T = \frac{12}{nK(K+1)} \sum_{k=1}^{K} R_k^2 - 3n(K+1) \tag{28}$$

where R_k is the sum of ranks for an algorithm. Using our data, the test statistic is:

$$\begin{aligned} T &= \frac{12}{5 \cdot 6(6+1)}(13^2 + 25^2 + 29^2 + 5^2 + 19^2 + 14^2) - 3 \cdot 5(6+1) \\ &= 21.7 \end{aligned} \tag{29}$$

The null hypothesis is rejected if $T \geq \chi^2_{K-1,1-\alpha}$ which is the $1-\alpha$ quantile of the Chi-squared distribution using $K-1$ degrees of freedom. Using $K-1 = 5$ degrees of freedom and $\alpha = 0.05$ we get $\chi^2_{5,0.95} = 11.07$. Since $21.7 > 11.07$ we reject the null hypothesis and claim that there is a statistically significant difference between the algorithms.

We now wish to see if any pairs of algorithms are statistically different. We test this by comparing all pairs as described in Section 4.2.2. Table 8 shows the results of the analysis. The null hypothesis for each pair is that the algorithms perform the same. We reject this hypothesis for the pairs shown in bold. Thus, we conclude that:

- (1+1)EA is statistically superior to UMDA and EHBSA
- EHBSA/WT is statistically superior to UMDA, EHBSA, and Mallows Model
- MallowsModel is statistically superior to EHBSA
- GeneralizedMallowsModel is statistically superior to UMDA and EHBSA

Algorithm	Problem Instance				
	5	**10**	**15**	**20**	**25**
(1+1)EA	70 (5)	605 (5)	1817 (5)	3720 (5)	5112 (5)
UMDA	221 (5)	1114 (5)	7930 (5)		
EHBSA	744 (5)	50502 (5)			
EHBSA/WT	46 (5)	248 (5)	577 (5)	1357 (5)	1989 (5)
MallowsModel					
GeneralizedMallowsModel	3 (1)				

Table 3: Fitness evaluations to find identity with population size $\sqrt{n}$

Algorithm	Problem Instance				
	5	**10**	**15**	**20**	**25**
(1+1)EA	90 (5)	694 (5)	1980 (5)	2534 (5)	5792 (5)
UMDA	136 (5)	3792 (5)	8736 (5)	9920 (1)	
EHBSA	882 (5)	31303 (5)			
EHBSA/WT	42 (5)	352 (5)	1214 (5)	2554 (1)	
MallowsModel	11 (2)	1270 (1)	5622 (2)		
GeneralizedMallowsModel	13 (3)	85 (3)			

Table 4: Fitness evaluations to find identity with population size n

Algorithm	Problem Instance				
	5	**10**	**15**	**20**	**25**
(1+1)EA	81 (5)	367 (5)	1976 (5)	3862 (5)	5230 (5)
UMDA	400 (5)	4480 (5)			
EHBSA	230 (5)				
EHBSA/WT	160 (5)	1330 (1)			
MallowsModel	89 (5)	645 (2)	1789 (1)	2588 (1)	
GeneralizedMallowsModel	79 (5)	456 (5)	1491 (5)	2747 (5)	4483 (5)

Table 5: Fitness evaluations to find identity with population size $10n$

Algorithm	Problem Instance				
	5	**10**	**15**	**20**	**25**
(1+1)EA	70	367	1817	2534	5112
UMDA	136	1114	7930	9920	
EHBSA	230	31303			
EHBSA/WT	42	248	577	1357	1989
MallowsModel	89	645	1789	2588	
GeneralizedMallowsModel	79	457	1491	2747	4483

Table 6: Best average number of fitness evaluations to find identity of each algorithm from all three population sizes

Algorithm	**Problem Instance**					
	5	**10**	**15**	**20**	**25**	**Sum of ranks**
(1+1)EA	2	2	4	2	3	**13**
UMDA	5	5	5	5	5	**25**
EHBSA	6	6	6	6	5	**29**
EHBSA/WT	1	1	1	1	1	**5**
MallowsModel	4	4	3	3	5	**19**
GeneralizedMallowsModel	3	3	2	4	2	**14**

Table 7: Algorithms ranked by best average number of fitness evaluations to find identity

Algorithm 1	**Algorithm 2**	**p value**
(1+1)EA	UMDA	**0.02126**
(1+1)EA	EHBSA	**0.00342**
(1+1)EA	EHBSA/WT	0.08815
(1+1)EA	MallowsModel	0.15525
(1+1)EA	GeneralizedMallowsModel	0.43289
UMDA	EHBSA	0.24948
UMDA	EHBSA/WT	**0.00036**
UMDA	MallowsModel	0.15525
UMDA	GeneralizedMallowsModel	**0.03149**
EHBSA	EHBSA/WT	**0.00002**
EHBSA	MallowsModel	**0.04548**
EHBSA	GeneralizedMallowsModel	**0.00561**
EHBSA/WT	MallowsModel	**0.00898**
EHBSA/WT	GeneralizedMallowsModel	0.06410
MallowsModel	GeneralizedMallowsModel	0.19901

Table 8: Pairwise comparison of algorithms for the sorting problem

Algorithm		**Problem Instance**				
		ulysses22	**att48**	**berlin52**	**st70**	**bier127**
(1+1)EA	Avg	83	45,445	10,288	1,162	250,685
	Std. Dev	7	2,293	440	83	3,046
UMDA	Avg	83	98,674	21,108	2,967	558,658
	Std. Dev	1	5,254	983	55	9,197
EHBSA	Avg	76	58,545	15,090	2,715	603,587
	Std. Dev	0	984	683	170	8,735
EHBSA/WT	Avg	76	71,011	20,049	3,085	590,968
	Std. Dev	0	3,508	648	34	6,197
MallowsModel	Avg	134	112,124	21,334	2,782	571,136
	Std. Dev	11	4,575	1,040	90	6,940
GeneralizedMallowsModel	Avg	131	117,665	21,303	2,716	560,210
	Std. Dev	13	5,025	347	177	9,198

Table 9: Average TSP solution found with population size $\sqrt{n}$

4.5 Solving TSP instances

To assess the performance of the algorithms on a real problem known to be NP-hard, we tested the algorithms on five instances of TSP from TSPlib [16]. Again, we used population sizes of $\sqrt{n}$, n, and $10n$. The results are shown in Tables 9, 10, and 11. Similar to the how we dealt with the sorting problem, we will take the best average for each algorithm for each problem instance as shown in Table 12. We then rank the algorithms as shown in Table 13.

The Friedman's test statistic is:

$$T = \frac{12}{5 \cdot 6(6+1)} \cdot (49 + 484 + 182.25 + 272.25 + 529 + 529) - 3 \cdot 5 \cdot (6+1) \quad = 11.9 \tag{30}$$

Since $11.9 > 11.07$, we reject the null hypothesis and conclude that the algorithms do not perform the same. We then find pairwise differences as shown in Table 14. We see that (1+1)EA is statistically superior to UMDA, MallowsModel, and GeneralizedMallowsModel. However, we cannot reject the null hypothesis between any of the EDAs.

5 Conclusion

To conclude, we have implemented and evaluated a number of Estimation of Distribution Algorithms for permutation based optimization problems. We analysed how different distance measures affect the performance of EDAs. While we can see differences and patterns, these were not statistically significant. On the sorting problem, we found that EHBSA/WT and Generalized Mallows Model perform statistically better than other EDAs. However, we were unable to show statistical differences between EDAs on TSP.

Future work includes extending the Generalized Mallows Model implementation to the Kernels of Mallows Model.

Algorithm		**Problem Instance**				
		ulysses22	**att48**	**berlin52**	**st70**	**bier127**
(1+1)EA	Avg	77	49,194	10,433	1,116	248,824
	Std. Dev	1	2,396	416	55	9,557
UMDA	Avg	81	96,025	21,689	2,909	565,022
	Std. Dev	7	6,094	619	84	11,278
EHBSA	Avg	76	135,834	25,832	3,252	578,293
	Std. Dev	0	1,847	611	34	3,848
EHBSA/WT	Avg	78	132,869	26,102	3,279	567,187
	Std. Dev	2	1,421	1,113	93	11,950
MallowsModel	Avg	119	116,223	24,088	3,038	570,615
	Std. Dev	6	3,206	349	26	8,318
GeneralizedMallowsModel	Avg	108	113,346	23,154	2,987	556,200
	Std. Dev	6	2,313	787	106	8,227

Table 10: Average TSP solution found with population size n

Algorithm		**Problem Instance**				
		ulysses22	**att48**	**berlin52**	**st70**	**bier127**
(1+1)EA	Avg	80	46,600	10,472	1,182	258,602
	Std. Dev	6	2,619	1,123	50	6,394
UMDA	Avg	104	111,565	23,709	2,987	557,777
	Std. Dev	8	3,402	640	45	8,113
EHBSA	Avg	136	123,800	25,048	3,026	561,435
	Std. Dev	8	3,330	373	75	5,114
EHBSA/WT	Avg	138	128,024	24,661	3,115	553,345
	Std. Dev	4	2,414	375	42	7,845
MallowsModel	Avg	114	115,903	23,760	3,061	551,994
	Std. Dev	5	2,504	213	50	9,038
GeneralizedMallowsModel	Avg	112	115,840	23,083	3,040	560,867
	Std. Dev	6	2,550	872	28	3,017

Table 11: Average TSP solution found with population size $10n$

Algorithm	**Problem Instance**				
	ulysses22	**att48**	**berlin52**	**st70**	**bier127**
(1+1)EA	77	45445	10288	1116	248824
UMDA	81	96025	21108	2909	557777
EHBSA	76	58545	15090	2715	561435
EHBSA/WT	76	71011	20049	3085	553345
MallowsModel	114	112124	21334	2782	551994
GeneralizedMallowsModel	108	113346	21303	2716	556200

Table 12: Best average solution for TSP of each algorithm from all three population sizes

Algorithm	**Problem Instance**					**Sum of ranks**
	ulysses22	**att48**	**berlin52**	**st70**	**bier127**	
(1+1)EA	3	1	1	1	1	7
UMDA	4	4	4	5	5	22
EHBSA	1.5	2	2	2	6	13.5
EHBSA/WT	1.5	3	3	6	3	16.5
MallowsModel	6	5	6	4	2	23
GeneralizedMallowsModel	5	6	5	3	4	23

Table 13: Algorithms ranked by best average solution for TSP

Algorithm 1	**Algorithm 2**	**p value**
(1+1)EA	UMDA	**0.00561**
(1+1)EA	EHBSA	0.13595
(1+1)EA	EHBSA/WT	0.05416
(1+1)EA	MallowsModel	**0.00342**
(1+1)EA	GeneralizedMallowsModel	**0.00342**
UMDA	EHBSA	0.07539
UMDA	EHBSA/WT	0.17627
UMDA	MallowsModel	0.43289
UMDA	GeneralizedMallowsModel	0.43289
EHBSA	EHBSA/WT	0.30604
EHBSA	MallowsModel	0.05416
EHBSA	GeneralizedMallowsModel	0.05416
EHBSA/WT	MallowsModel	0.13595
EHBSA/WT	GeneralizedMallowsModel	0.13595
MallowsModel	GeneralizedMallowsModel	0.50000

Table 14: Pairwise compairson of algorithms for solving TSP

References

[1] Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and Jose A. Lozano. A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. *Progress in Artificial Intelligence*, 1(1):103–117, 2012.

[2] Martin Zaefferer, Jörg Stork, and Thomas Bartz Beielstein. Distance measures for permutations in combinatorial efficient global optimization. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8672:373–383, 2014.

[3] Michael Deza, Liens-ecole Normale Supérieure, and Tayuan Huang. Metrics on permutations, a survey. 2012.

[4] M. A. Fligner and J. S. Verducci. Distance based ranking models. *Journal of the Royal Statistical Society. Series B (methodological)*, 48(3):359–369, 1986.

[5] Marina Meila, Kapil Phadnis, Arthur Patterson, and Jeff A. Bilmes. Consensus ranking under the exponential model. 2012.

[6] Jens Scharnow, Karsten Tinnefeld, and Ingo Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3(4):349–366, 2004.

[7] Heinz Muehlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1997.

[8] Shigeyoshi Tsutsui, Martin Pelikan, and David E. Goldberg. Using edge histogram models to solve permutation problems with probabilistic model-building genetic algorithms. Technical report, 2003.

[9] Josu Ceberio. Solving permutation problems with estimation of distribution algorithms and extensions thereof. 2014.

[10] Josu Ceberio, Alexander Mendiburu, and Jose A. Lozano. Introducing the mallows model on estimation of distribution algorithms. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7063 LNCS(PART 2):461–470, 2011.

[11] Michel Truchon. Borda and the maximum likelihood approach to vote aggregation. *Mathematical Social Sciences*, 55(1):96–102, 2008.

[12] scipy.optimize.newton. url: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.newton.html last accessed: 2020-05-23.

[13] Martin S. Krejca and Carsten Witt. Theory of estimation-of-distribution algorithms. *Natural Computing Series*, pages 405–442, 2020.

[14] Dulce G. Pereira, Anabela Afonso, and Fátima Melo Medeiros. Overview of friedmans test and post-hoc analysis. *Communications in Statistics: Simulation and Computation*, 44(10):2636–2653, 2015.

[15] Salvador García, Francisco Herrera, and John Shawe-taylor. An extension on —statistical comparisons of classifiers over multiple data sets for all pairwise comparisons. 2012.

[16] Mp-testdata - the tsplib symmetric traveling salesman problem instances. url: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html. last accesed: 2020-05-26.