On the Definition of Dynamic Permutation Problems under Landscape Rotation

Joan Alza
The Robert Gordon University
Aberdeen, Scotland
j.alza-santos@rgu.ac.uk

Mark Bartlett
The Robert Gordon University
Aberdeen, Scotland
m.bartlett3@rgu.ac.uk

Josu Ceberio
University of the Basque Country (UPV/EHU)
Donostia, Spain
josu.ceberio@ehu.eus

John McCall
The Robert Gordon University
Aberdeen, Scotland
j.mccall@rgu.ac.uk

ABSTRACT

Dynamic optimisation problems (DOPs) are optimisation problems that change over time. Typically, DOPs have been defined as a sequence of static problems, and the dynamism has been inserted into existing static problems using different techniques. In the case of dynamic permutation problems, this process has been usually done by the rotation of the landscape. This technique modifies the encoding of the problem and maintains its structure over time. Commonly, the changes are performed based on the previous state, recreating a concatenated changing problem. However, despite its simplicity, our intuition is that, in general, the landscape rotation may induce severe changes that lead to problems whose resemblance to the previous state is limited, if not null. Therefore, the problem should not be classified as a DOP, but as a sequence of unrelated problems. In order to test this, we consider the flow shop scheduling problem (FSSP) as a case study and the rotation technique that relabels the encoding of the problem according to a permutation. We compare the performance of two versions of the state-of-the-art algorithm for that problem on a wide experimental study: an adaptive version that benefits from the previous knowledge and a restarting version. Conducted experiments confirm our intuition and reveal that, surprisingly, it is preferable to restart the search when the problem changes even for some slight rotations. Consequently, the use of the rotation technique to recreate dynamic permutation problems is revealed in this work.

CCS CONCEPTS

• **Mathematics of computing** → *Permutations and combinations*; *Evolutionary algorithms*; • **Theory of computation** → *Evolutionary algorithms*;

KEYWORDS

Dynamic Optimization Problem, Flow Shop Scheduling Problem, Permutation Problem, Benchmark Generator, Evolutionary Computation

ACM Reference format:
Joan Alza, Mark Bartlett, Josu Ceberio, and John McCall. 2019. On the Definition of Dynamic Permutation Problems under Landscape Rotation. In *Proceedings of Genetic and Evolutionary Computation Conference Companion, Prague, Czech Republic, July 13–17, 2019 (GECCO '19 Companion),* 9 pages.
https://doi.org/10.1145/3319619.3326840

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3326840

1 INTRODUCTION

In real-world situations, optimisation problems usually depend on changing conditions, and it is important for an optimisation algorithm to react efficiently. Known as dynamic optimisation problems (DOPs), these problems rely on solving an optimisation problem by an optimisation algorithm in a time window in which the problem changes at least once. Consequently, the algorithm requires a reaction to the change to provide new promising solutions. Examples of real-world dynamic optimisation problems include the arrival of a new task or the improper functioning of a machine in a scheduling problem, the influence of traffic in the transportation domain, or customer demand in logistics.

In the literature, many definitions have been proposed to refer to DOPs. Some researchers have defined DOPs as a *sequence of static problems linked up by a dynamic rule* [12, 16], whereas others define them as *optimisation problems composed by time-dependent parameters* [3, 5]. Not limited to that, in [11], the author inserted the way that the algorithm solves the problem defining DOPs as *a special class of dynamic problems that are solved online by an optimisation algorithm as time goes by*. However, it is our belief that not any sequence of static problems should be considered a DOP. Consider a case where a severe problem-change translates to a problem completely different. In that case, the adaptability inserted to existing evolutionary algorithms is harmful to solve the problem effectively.

Related to the definition of DOPs, the simulation of realistic scenarios has been an outstanding issue due to the recreation of real-world applications in academic research. In short, DOPs modify the problem over time hindering the search for the new global optimum. In the literature, some dynamic benchmark generators such

as Moving Peaks [4] or exclusive-or (XOR) [14] have been presented. The XOR DOP generator has been widely used since it rotates the landscape at each change-step (moment of change), maintaining the structure of the problem. Although this technique was initially proposed to generate dynamism into binary problems, later works extend it to other fields such as permutation problems [10]. It appears a useful technique to draw preliminary conclusions because it is a quick and straightforward way to insert dynamism into an existing permutation problem. However, in this manuscript, we put in question the utility of this technique to generate dynamism since it might create such severe modifications on the problem that the scenario after a change might be categorised as a new problem completely different.

To that end, we consider the flow shop scheduling problem (FSSP) as a case study and the state-of-the-art algorithm for that problem, the random key based estimation of distribution algorithm (RKEDA) [2]. We select the first instances (considering the different number of machines) from the original instances of the FSSP [13] for the sizes 20, 50 and 100, and we insert dynamism considering different change-severity (magnitude of change) values. The landscape rotations are produced by three distance metrics for permutation space [7]: Kendall's-τ, Cayley and Ulam.

For the sake of solving dynamic FSSPs, two different versions of the RKEDA are presented. The first version adjusts to a problem change while retaining some previously earned knowledge (aRKEDA). The second version restarts the search process by generating an entirely new population every change-step (rRKEDA). The aim is to compare the performance of aRKEDA and rRKEDA through different change-severity values under the landscape rotation. Our hypothesis is that reusing knowledge from the previous state at severe problem-changes will be useless, if not harmful. Therefore, restarting the algorithm will be more beneficial in that case. At this point, we can conclude that it is not appropriate to call dynamic to the changing problem.

Our experiments, surprisingly, show the few cases where the adaptation using previous knowledge is effective. As expected, for low-severity changes aRKEDA performs better than rRKEDA, but they tend to reverse their role so rapidly as change-severity increases that restarting the algorithm from scratch is preferable even in some of the slightly changing problems. In this way, a sequence of FSSPs linked up by the landscape rotation should not be considered a DOP but a sequence of problems completely different in case that the effectiveness of restarting the algorithm is higher than adapting to problem-changes.

The rest of this paper is organised as follows. Section 2 presents the context used in this work by explaining the background of permutations. Section 3 describes the commonly used definitions for DOPs, our points to consider and the dynamic benchmark generator used. Section 4 introduces the RKEDA, and the methods used to deal with the dynamism. Section 5 explores the results obtained by both techniques, before Section 6 provides discussion about the interpretation of the results. Finally, Section 7 concludes the paper by giving some conclusions and future works.

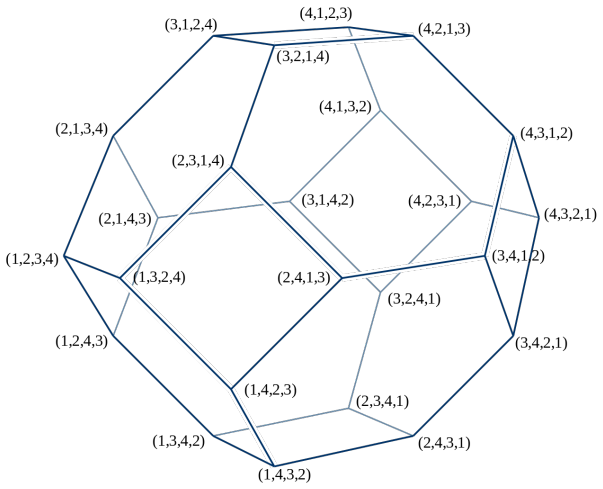

Figure 1: Permutahedron[1] of order 4.

2 PERMUTATION SPACE

A permutation is a bijection from the set S, composed of natural numbers, to itself. For a set of size n, there are $n!$ possible permutations, and they form the algebraic group called *symmetric group*, denoted as $\mathbb{S}_n$. Figure 1 displays a graphical representation of $\mathbb{S}_n$ of size 4 in the form of a Permutahedron[1]. Commonly permutations are represented as $\sigma, \pi \in \mathbb{S}_n$, where $\sigma(i)$ stands for the element at position i.

There is a special permutation, the identity permutation, in which the item i is mapped on the position i, and it is usually denoted as e. In addition, for every permutation σ, there is an inverse permutation $\sigma^{-1} \in \mathbb{S}_n$, where each item and its position on the permutation σ are exchanged.

The composition of two permutations is used to provide a new permutation, and it is defined as $\sigma \circ \pi\ (i) = \sigma(\pi(i))$. In this sense, the composition of permutation σ and its inverse permutation σ^{-1} is equal to the identity permutation, $\sigma \circ \sigma^{-1} = e$. It is worth noting that the composition of two permutations is not in general commutative, $\sigma \circ \pi \neq \pi \circ \sigma$.

2.1 Distance metrics

The distance between permutations can be defined as the minimum number of steps to change one permutation into another. There are many metrics, although Kendall's-τ, Cayley and Ulam distances have been prominently used as the distance between permutations in combinatorial space [7].

Given two permutations σ and π, Kendall's-τ metric counts the minimum number of pairwise disagreements between two permutations. Equivalently, it corresponds to the number of adjacent swaps to turn σ^{-1} into π^{-1}. The maximum distance between two permutations under Kendall's-τ metric is $\binom{n}{2}$, where n represents the size of the permutations.

The Cayley metric counts the minimum number of (possibly non-adjacent) swaps that are needed to turn σ into π. In the case of

[1] https://upload.wikimedia.org/wikipedia/commons/3/3e/Permutohedron.svg

Cayley metric, the maximum distance between two permutations is $n-1$.

Finally, the Ulam metric represents the minimum number of insertions needed to transform a permutation into another. The maximum Ulam distance between two permutations is $n-1$.

For a better understanding about the metrics, consider the following two permutations: $\sigma = (3,4,2,1)$ and $e = (1,2,3,4)$, whose inverts permutations are $\sigma^{-1} = (4,3,1,2)$ and $\pi^{-1} = (1,2,3,4)$. The distance between both permutations is the following for each metric:

- The Kendall's-τ distance between σ^{-1} and π^{-1} is equal to 5: $(4,3,1,2) \xrightarrow{3-1} (4,1,3,2) \xrightarrow{4-1} (1,4,3,2) \xrightarrow{3-2} (1,4,2,3) \xrightarrow{4-2} (1,2,4,3) \xrightarrow{4-3} (1,2,3,4)$.
- The Cayley distance between σ and π is equal to 3: $(3,4,2,1) \xrightarrow{3-1} (1,4,2,3) \xrightarrow{4-2} (1,2,4,3) \xrightarrow{4-3} (1,2,3,4)$.
- The Ulam distance between σ and π is equal to 2: $(3,4,2,1) \xrightarrow{1} (1,3,4,2) \xrightarrow{2} (1,2,3,4)$.

3 DYNAMIC OPTIMISATION PROBLEMS

In the literature, researchers have defined DOPs as problems that change over time while an optimisation algorithm solves them. Changes may affect the objective function, the problem instance, or the constraints of the problem, among other elements. They are divided into two types: dimensional changes and non-dimensional changes. Dimensional changes alter the cardinality of the solution space while non-dimensional ones change the variables and/or constraints of the problems. Dynamic problems with dimensional changes tend to be harder to solve [9].

Recently, researchers have defined DOPs in several ways: (i) a sequence of static problems linked up by a dynamic rule, (ii) optimisation problems composed by time-dependent parameters or (iii) dynamic problems that are solved by an optimisation algorithm as time goes by. In the latter case [11], the author proposed a definition composed of *full-description forms*, *dynamic drivers* and an *optimisation algorithm*. That work interprets DOPs as *full-description forms* that represent a finite set of static instances governed by *dynamic drivers* while an optimisation algorithm solves the problem *online*[2]. Furthermore, others have also mentioned that the definition of a *sequence of static problems linked up by a dynamic rule* might be ambiguous because of the way to cross from a static problem into another [11]. Dynamic drivers on the previously introduced definition addressed that problem. For further details, we refer the interested reader to Chapter 4 in [11].

3.1 Dynamic benchmark generator for combinatorial space

Dynamic benchmark generators are tools to construct DOPs using controllable parameters such as the change-frequency, the change-severity or the number of changes. Often, DOPs are considered hard to construct because of the difficulty of simulating real-world situations. Therefore, DOPs are treated mainly as academic problems. Usually, they mix real-world data and randomly generated data in order to reduce the evaluation complexity of real-world data [16]. Hence, most of the proposed dynamic benchmark generators have focused on empirical testing.

The XOR DOP generator [14] is one of the most popular generators because it constructs DOPs from any static binary problem. The generator rotates the landscape at a change-step applying an exclusive-or operator to every individual of the search space, shifting them to a new location of the search space. In [10], the authors extended the previous method to the permutation space modifying the encoding of the problem instance. In any case, the structure of the problem is maintained over time without affecting the fitness of the global optimum.

These techniques offer a quick and straightforward way to generate dynamism in any combinatorial problem, but it is not likely to find them in real-world situations.

In this work, DOPs are obtained from a dynamic benchmark generator similar to the one presented in [10]. As we are working with dynamic permutation-based optimisation problems, we generate different dynamic scenarios to recreate changes on a static permutation problem. In this way, we change the labelling of the previous scenario to rotate the landscape, creating a concatenated changing problem. To that end, we compose the solutions of the problem with a random permutation at a specific distance value using the metrics presented in Section 2, where the increment of the distance produces more severe changes in the problem. Mathematically, given an objective function f and a solution $\sigma \in \mathbb{S}_n$,

$$f(e \circ \sigma) = f(\sigma) \xrightarrow{c_1} f(\pi_1 \circ \sigma) \xrightarrow{c_2} f(\pi_2 \circ \pi_1 \circ \sigma) \xrightarrow{c_3} \cdots \xrightarrow{c_k} f(\Omega \circ \sigma), \tag{1}$$

where e is the identity permutation, c_i is the i^{th} change of the problem, k is the number of changes and $\Omega = (\pi_k \circ \cdots \circ \pi_1)$ is the composition of the previous permutations.

In this way, the dynamic benchmark generator reorders the labels of the items that compose the problem according to a permutation.

4 CASE STUDY: FSSP AND RKEDA

In the flow shop scheduling problem (FSSP) [1], the permutation represents a set of n jobs that have to be scheduled on m machines reducing a given measurement. Regardless of the measurement used, the general idea is to maintain the flow of the sequence of jobs minimising the idle and waiting time of machines. FSSP is categorised as a *NP-hard* problem [6].

Typically, in optimisation, two different measurements are considered: makespan and the total flow time measurements. The makespan measurement counts the processing time taken to finish a batch of jobs. In contrast, the total flow time measurement sums the time to complete each job. The mathematical representation of the FSSP using the total flow time measurement is defined as:

$$F(\sigma) = \sum_{i \in n} C_{\sigma(i),m},$$

where $F(\sigma)$ is the fitness value of permutation σ and C is the completion time of each job at each machine. The completion time of

[2]We return to this point in Section 6

Algorithm 1 Pseudo-code of both RKEDA

1: $\boldsymbol{P} \leftarrow$ Generate M random individuals.
2: $\theta \leftarrow$ Initialise cooling scheme.
3: Evaluate, sort and normalise individuals in $\boldsymbol{P}$.
4: $best \leftarrow P_{best}$.
5: **repeat**
6: **if** $best = P_{best} \wedge f(best) != f(P_{best})$ **then**
7: **if** rRKEDA **then**
8: Back to line 1.
9: **else if** aRKEDA **then**
10: $\theta \leftarrow$ Initialise cooling scheme.
11: **end if**
12: **end if**
13: $\boldsymbol{P}_{top} \leftarrow$ Select best t_s solutions from $\boldsymbol{P}$.
14: $\boldsymbol{\mu} \leftarrow$ Average of $\boldsymbol{P}_{top}$ for each position.
15: $\boldsymbol{P'} \leftarrow$ Create new population of size M by sampling $N(\boldsymbol{\mu}, \theta)$.
16: $\boldsymbol{P} \leftarrow \boldsymbol{P'}$.
17: $\theta \leftarrow$ Update cooling scheme.
18: Evaluate, sort and normalise individuals in $\boldsymbol{P}$.
19: $best \leftarrow P_{best}$.
20: **until** Stopping criteria is met.

job i on machine j may be calculated recursively as follows.

$$C_{i,j} = \begin{cases} p_{i,j}, & \text{if } i,j = 1, \\ p_{i,j} + C_{i,j-1}, & \text{if } i = 1, j > 1, \\ p_{i,j} + C_{i-1,j}, & \text{if } i > 1, j = 1, \\ p_{i,j} + \max\{C_{i-1,j}, C_{i,j-1}\}, & \text{if } i > 1, j > 1, \end{cases}$$

where $p_{i,j}$ is the processing time of job i at machine j.

The fitness value of a solution depends on the position of each item in addition to the whole ordering of the permutation. Note that there is a strong correlation between a job and the ordering of the rest of the jobs.

In order to solve the FSSP effectively, the random key based EDA (RKEDA) has been proposed [2]. RKEDA is an EDA that creates new solutions by learning and sampling a Gaussian distribution based on mean values of the promising solutions of the population and a variance parameter. The variance parameter, like that used in *simulated annealing*, is based on a gradient descent cooling that decreases the probability of accepting worse solutions as the solution space is explored. In this way, it is cooled from a set initial value until it reaches zero at the end of the process.

As we are working with DOPs, it is necessary to modify the algorithm by introducing an optimisation approach to react to problem changes. Commonly, the consciousness of changes has been assumed just using few detectors [15].In this paper, we adopt a simple scheme where the algorithm realises a change has occurred by the variation of the fitness value of the solutions in the population with respect to the previous generation whereas their permutation structure remains unchanged.

Two different versions are utilised in our study to analyse the resemblance between scenarios: an adaptive version and an iterative version of the RKEDA. The adaptive version of the RKEDA (aRKEDA) maintains the knowledge earned before a problem-change to adjust to the new state. The idea is to benefit from the new environment transferring the knowledge of the previous state.

In contrast, the restarting version of the RKEDA (rRKEDA) restarts the optimisation process of the algorithm from scratch when a change occurs, generating a new random population. In this case, the new scenario is considered as a new problem.

The purpose of comparing both version is to analyse whether a problem should be considered a DOP or, in contrast, a sequence of unrelated static problems. Therefore, a sequence of static problems should only be considered a DOP in the event that aRKEDA outperforms the performance of rRKEDA.

Algorithm 1 displays the pseudo-code of both versions. When the problem changes, aRKEDA only resets the variance whereas rRKEDA restarts the optimisation process from scratch.

5 EXPERIMENTAL STUDY

In this section, we present the experimental design and the results obtained. The section is divided into three parts. First, we explain the dynamism inserted to existing problems. Next, we present the experimental setup to run the experiments. Finally, we present the results obtained.

5.1 Dynamism generation

As previously mentioned, our DOPs are concatenated scenarios generated by a rotation technique. Initially, the rotation is applied into an existing instance to generate a new scenario, and the next changes are performed based on the previously generated scenarios. The number of problem-changes, the change-severity and the metric to calculate the change-severity are aspects to be borne in mind to generate DOPs. For each DOP, we considered 10 changes that occur regularly and are distributed periodically through the optimisation process. All the changes of each DOP share the same change-severity, so the changes perturb similarly every change-step.

In Section 2.1, we presented three different metrics to perform landscape rotations: Kendall's-τ distance, Cayley distance and Ulam distance. It is worth mentioning that Cayley and Ulam distances share the maximum distance between permutations, $n-1$, whereas Kendall's-τ metric extends to $\binom{n}{2}$. It is computationally too expensive to test all the cases for the Kendall's-τ metric, so we have established limitations to reduce the computational cost limiting the information loss:

- On problems of size 20, generate all the combinations.
- On problems of size 50, generate DOPs considering the Kendall's-τ distance from 1 to 150.
- On problems of size 100, generate DOPs considering the Kendall's-τ distance from 1 to 50.

All metrics follow the same pattern for modifying the encoding of the problem. A permutation at the required distance is generated uniformly at random and composed with the permutation of the previous state (see Equation 1 at Section 3.1). The R package `PerMallows` [8] is used for this process.

Table 1: Parameter values for the experimental study.

Parameter	Default value
Population size	$10n$
Truncation size	10%
Elitism criteria	TRUE
Initial variance	0.15
Number of samples	$10n$
Stopping criteria	$100nk$ generations

n: problem size
k: number of changes

5.2 Experimental setup

Both algorithms are executed over many dynamic benchmarks generated from the static versions of the FSSP. The original Taillard's instances [13] are used as the first scenario of each dynamic benchmark. The configurations used in this work are as follows[3].

- 20×5, 20×10 and 20×20.
- 50×5, 50×10 and 50×20.
- 100×5, 100×10 and 100×20.

In this way, for each instance, we generated 30 different DOPs with each distance metric considering the limitations presented in Section 5.1.

The execution parameters for the RKEDA algorithms are those from [2], slightly modified to account for the dynamic behaviour, as shown in Table 1. The elitism criteria is used to detect a problem-change by the variation of the fitness of the best solution of the population whereas the permutation is maintained.

A basic scheme [5] is used to calculate the performance of the algorithms for every dynamic benchmark: the average relative percentage deviation (ARPD). As the structure of the problem is maintained through the optimisation process, the best-known value is known permanently allowing the usage of this performance measure. In this case, the ARPD is used to minimise the sum of the distance from the optimum to the best solution at each generation, so it measures the quality of the best fitness value at each generation. It is calculated as follow:

$$ARPD = \frac{1}{G} \sum_{i \in G} \frac{f_i(best) - \text{Best known}}{\text{Best known}},$$

where G is the maximum number of generations and $f_i(best)$ is the fitness of the best solution at generation i.

5.3 Results

The results[4] are summarised in Table 2. The table shows the number of times in which aRKEDA outperformed rRKEDA in terms of the ARPD measure and the percentage of rotation distances for which the generated problem can be considered as a DOP under the Cayley, Kendall's-τ and Ulam metrics. In general, increasing the problem size increases the number of times that the adaptive version is preferable to restarting the process from scratch. These results suggest that most of the studied benchmarks are probably better viewed as sequences of unrelated static problems than as dynamic problems.

Table 2: Number of case that aRKEDA outperforms rRKEDA. The number in parentheses displays the percentage considering the maximum distance between permutations.

Jobs	Cayley			Kendall's-τ			Ulam		
20	2 (10%)	1 (5%)	2 (10%)	2 (1.05%)	3 (1.58%)	3 (1.58%)	0 (0%)	0 (0%)	0 (0%)
50	7 (14%)	7 (14%)	12 (26%)	14 (1.14%)	7 (0.57%)	8 (0.65%)	0 (0%)	2 (4%)	1 (2%)
100	19 (19%)	15 (15%)	20 (20%)	16 (0.32%)	19 (0.38%)	26 (0.52%)	-	-	-
	5	10	20	5	10	20	5	10	20
	Machines								

For a better understanding of the results, Figure 2, Figure 3, and Figure 4 show the distribution of the ARPD through the DOPs generated at different permutation distance changes for instances of size 20, 50 and 100. The plots confirm that, for small changes, it is beneficial to transfer the previous knowledge, but the situation reverses rapidly as the change-severity increases. However, in DOPs of size 20 generated by Ulam metric, it is always more effective to restart the algorithm from scratch when the problem changes. The increase of the problem size extends the preference of using aRKEDA for slight changes up to the point of being much more beneficial than rRKEDA on DOPs generated at the smallest distance. The continuity of the restarting version could be understood because the algorithm is able to obtain approximately the same fitness value at each change-period.

It is worth emphasising some aspects of aRKEDA on problems of size 20. On the one hand, the increase of the number of machines in the problem produces a chaotic behaviour of the adaptive version as can be observed in the variance of the results. On the other hand, the Kendall's-τ and Ulam metrics reflect a curious case on instances with 20 machines. Kendall's-τ metric reflects an arc shape where aRKEDA is preferred on DOPs at the minimum and maximum distances. Surprisingly, under the Ulam metric, aRKEDA is only preferred at the maximum distance, since at minimum distance it is even preferable to restart the algorithm.

Analysing the performance of aRKEDA and rRKEDA, we might conclude that the values in Table 2 are mostly distributed over DOPs that are generated at small distances, but it also includes the anomaly cases mentioned above. With respect to the performance of each metric, the Ulam metric shows that restarting the algorithm from scratch is almost always the best option.

6 DISCUSSION

The experiments conducted reveal that, under the rotation technique with the considered distances, restarting the optimisation is almost always preferable to reusing previously earned knowledge in the FSSP, even when slight modifications are applied. More than three-quarters of the problems generated under the rotation technique produce such severe modifications that considering the new state as a new problem is the best option. However, the performance difference between both versions is considerably higher for slightly rotating DOPs. Be that as it may, in most of the cases, the problem generated should not be considered a DOP, but a sequence of unrelated problems.

The percentages with respect to the maximum distances have indicative purposes only. The means of the percentage over each

[3]Number of jobs × Number of machines
[4]The Ulam metric experiments on problems of size 100 have not been performed because of the high computational cost to generate uniformly at random permutations

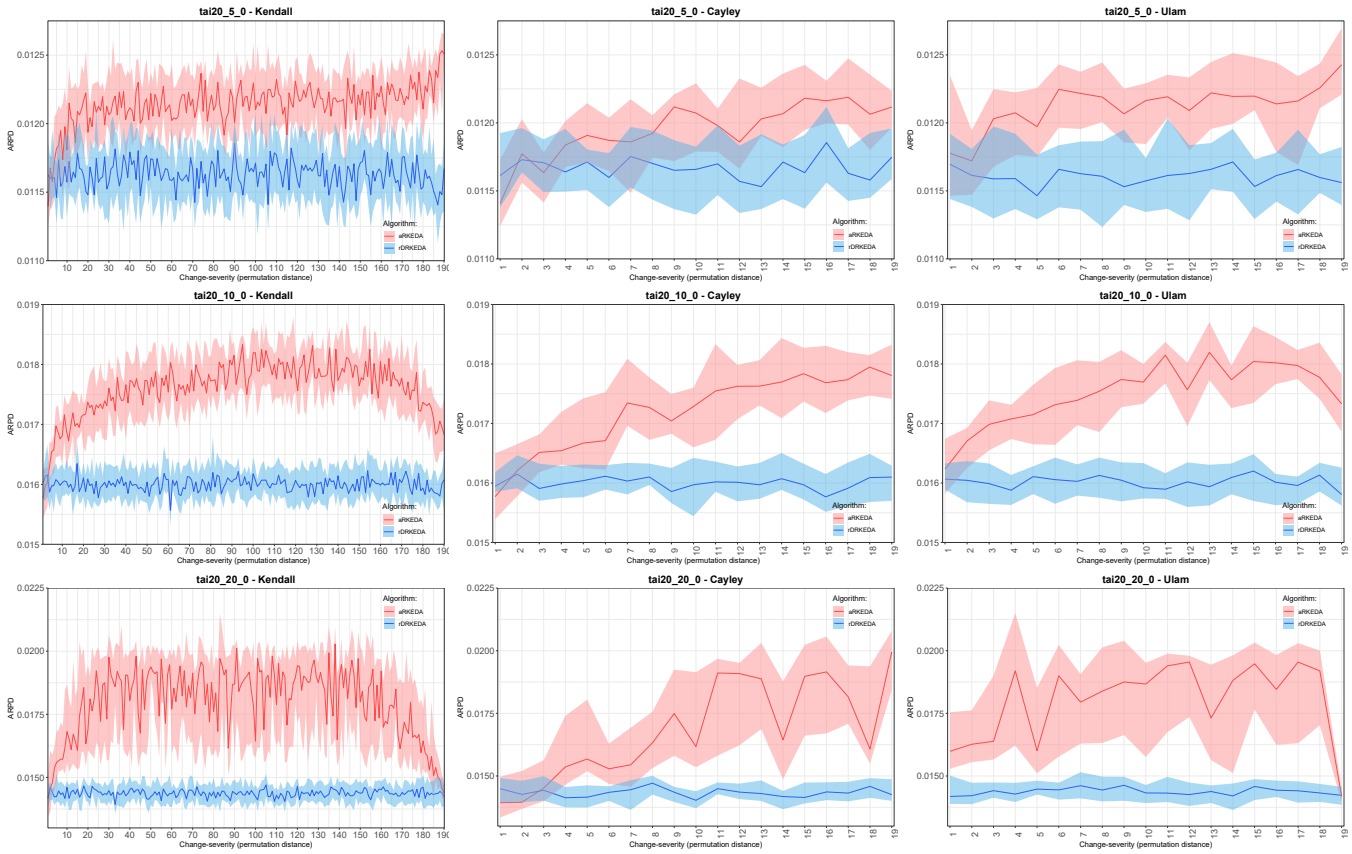

Figure 2: Performance of aRKEDA and rRKEDA for instances of size 20 showing the mean and interquartile ribbon of each run through different change-severities (in terms of permutation distances). The plots are distributed by columns specifying the metric used (Kendall's-τ, Cayley and Ulam from left to right) and by rows representing the number of machines of each problem (5, 10 and 20 machines from top to bottom).

metric are 15% on Cayley metric, 1% on Kendall's-τ metric and 0.5% on Ulam metric. It is of special interest that although the percentages are relatively small in all the cases, Kendall's-τ and Ulam metrics present surprisingly few cases in which reusing previous knowledge is beneficial.

In the case of Ulam metric, restarting the algorithm is almost always the best option, even for permutation distances equal to 1. It can be justified as the Ulam metric produces more chaotic changes compared to the other metrics. This is mainly motivated by the type of rotation (relabelling) applied by the Ulam metric. In contrast to Kendall's-τ and Cayley metrics, where only a few elements are swapped for slight changes, a simple insert operation on the Ulam metric might cause the relabelling of all the permutation. The following example will clarify the concept.

Let us consider the same example proposed in Section 2.1, where $\sigma = (3, 4, 2, 1)$ and $e = (1, 2, 3, 4)$. The simple movement of the item $\sigma(4)$ to the position 1 causes the translation of the items at the positions $\{1, 2, 3\}$ one position to the right. Therefore, all the item of the permutation are relabelled on a simple insertion, and it is likely that even a slight rotation produces a severe change on the fitness in the landscape of solutions.

As commented in Section 3.1, the rotation technique gives an obvious and straightforward way to generate dynamism in existing static permutation problems. Nonetheless, the results obtained for the case study in this work show that the rotation technique would generate a set of independent and unrelated problems that should be senseless to associated with DOPs.

Taking that fact into account, we glimpse the need to extend the existing definitions for dynamic permutation-based optimisation problems, especially concerning the change-severity to distinguish related-sequences of static problems from sequences of completely different problems. In this sense, the author in [11] uses notations such as full-description form, a dynamic driver and the optimisation algorithm to define DOPs that encompasses more aspects of dynamism. Therefore, the definition used by the author in [11] should be explored and enhanced in order to include the impact of the change on the performance of the algorithm for a more accurate description of dynamic permutation-based optimisation problems.

7 CONCLUSIONS AND FUTURE WORK

Over the last decades, DOPs have been a topic of growing interest in the field of evolutionary optimisation due to their similarity with

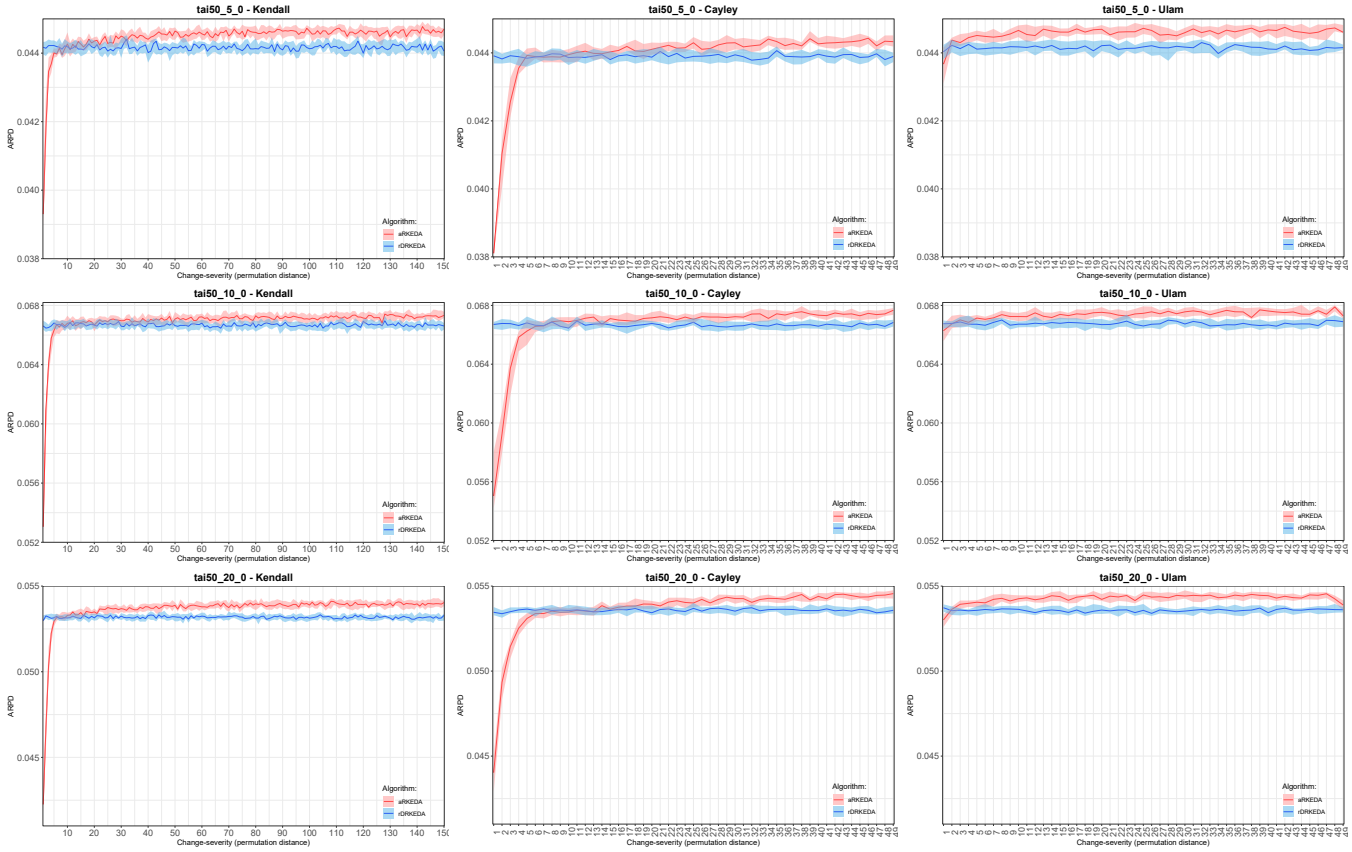

Figure 3: Performance of aRKEDA and rRKEDA for instances of size 50 showing the mean and interquartile ribbon of each run through different change-severity values (in terms of permutation distances). The plots are distributed by columns specifying the metric used (Kendall's-τ, Cayley and Ulam from left to right) and by rows representing the number of machines of each problem (5, 10 and 20 machines from top to bottom).

existing real-world situations where problems change over time. In academic research, DOPs have been defined in several ways, although our intuition is that existing definitions do not fully cover all aspects of dynamism. Current definitions of DOPs encompass all type of changing problem, including problems that change drastically, which, in our opinion, should not be interpreted as DOPs, but as sequences of different problems. In that case, considering the new state as a new problem might be more beneficial than adjusting an algorithm to overcome a change of the problem.

In this paper, we test the current definitions considering permutation problems under the landscape rotation technique that generates dynamism into an existing permutation problem. This technique shifts the landscape of solutions maintaining the structure of the problem. In order to illustrate the sense of rotating the landscape to generate dynamic problems, we considered the FSSP and two versions of its state-of-the-art algorithm to deal with dynamism: an adaptive version and a restarting version. The adaptive version stays ahead from the previous knowledge to address a problem-change, whereas the restarting version initialises the algorithm from a new random solution. The aim is to identify those rotations that, due to their severity, make the restarting version of the algorithm outperform the adaptive version. Therefore, the problem should not be considered a dynamic problem but a sequence of unrelated problems.

The results revealed that restarting the algorithm is preferable for the majority of tested landscape rotations because of the severity of the produced changes. Looking at the results over the metrics, we perceive that the adaptive version performed poorly for Ulam metric to the point of restarting the algorithm is almost always the best option. That could be understood by the relabelling type of the Ulam metric since a simple insertion might produce shifting all the elements of the permutation (a full relabelling). In contrast, the Kendall's-τ and the Cayley metrics only swap some items for a simple change, producing more limited changes that the ones produced by the Ulam metric.

This work can be extended in several ways. One of the most interesting research lines is to analyse and motivate the performance difference of the metrics used in this work such as analysing the reason for which generating dynamism under the Kendall's-τ and Cayley distances produces such different results. One appealing approach would be analysing the number of relabelled items for each metric and tracking the quality of the model. Not limited to that,

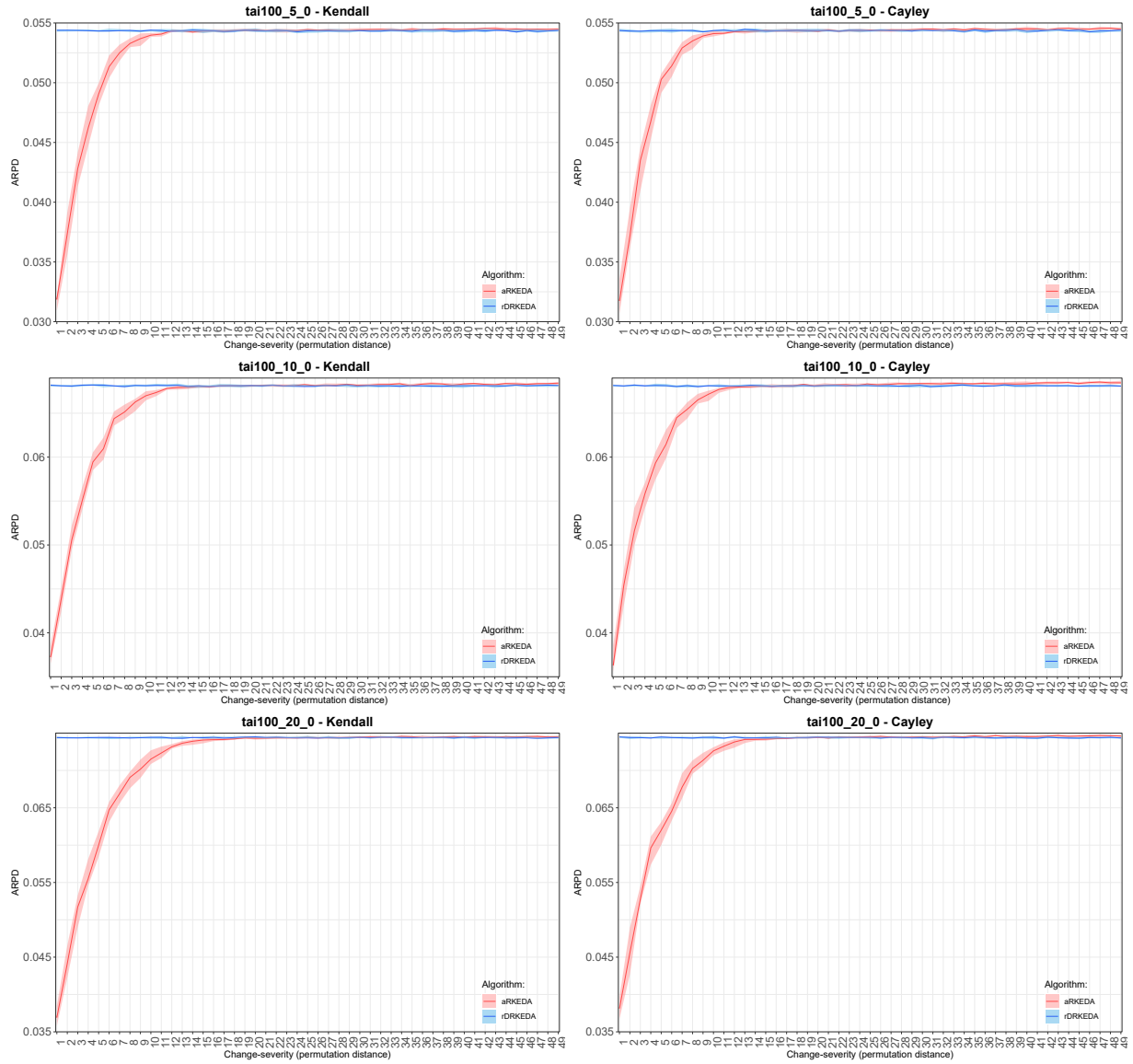

Figure 4: Performance of aRKEDA and rRKEDA for instances of size 100 showing the mean and interquartile ribbon of each run through different change-severity values (in terms of permutation distances). The plots are distributed by columns specifying the metric used (Kendall's-τ and Cayley) and by rows representing the number of machines of each problem (5, 10 and 20 machines from top to bottom).

the experimental comparison could be extended to other metrics such as the Hamming metric [7]. Other possible lines for future work could be considering other algorithms to analyse their behaviour since deciding whether a problem is dynamic or a sequence of unrelated problems has been carried out for a particular case. Similarly, other methods of generating dynamism, other permutation problems or even more ways to measure the performance of the algorithms could be also analysed in future investigations. Finally, we find it interesting to study the existing definitions for DOPs and, if necessary, extend them to include more observations such as the severity of the problem-change in their description.

ACKNOWLEDGMENTS

This work has been partially supported by the project TIN2016-78365-R from the Spanish Ministry of Economy, Industry and Competitiveness.

REFERENCES

[1] Ali Allahverdi, C.T. Daniel Ng, Tai Chiu Edwin Cheng, and Mikhail Y. Kovalyov. 2008. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research* 187, 3 (2008), 985 – 1032. https://doi.org/10.1016/j.ejor.2006.06.060

[2] Mayowa Ayodele, John McCall, Olivier Regnier-Coudert, and Liam Bowie. 2017. A Random Key based Estimation of Distribution Algorithm for the Permutation

Flowshop Scheduling Problem. In *2017 IEEE Congress on Evolutionary Computation (CEC)*. 2364–2371. https://doi.org/10.1109/CEC.2017.7969591

[3] Thomas Back. 1998. On the behavior of evolutionary algorithms in dynamic environments. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*. 446–451. https://doi.org/10.1109/ICEC.1998.699839

[4] Jurgen Branke. 1999. Memory enhanced evolutionary algorithms for changing optimization problems. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, Vol. 3. 1875–1882 Vol. 3. https://doi.org/10.1109/CEC.1999.785502

[5] Carlos Cruz, Juan R. González, and David A. Pelta. 2011. Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing* 15, 7 (01 Jul 2011), 1427–1448. https://doi.org/10.1007/s00500-010-0681-0

[6] Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

[7] Ekhine Irurozki. 2014. *Sampling and learning distance-based probability models for permutation spaces*. Ph.D. Dissertation.

[8] Ekhine Irurozki, Borja Calvo, and Jose Lozano. 2016. PerMallows: An R Package for Mallows and Generalized Mallows Models. *Journal of Statistical Software, Articles* 71, 12 (2016), 1–30. https://doi.org/10.18637/jss.v071.i12

[9] Changhe Li and Shengxiang Yang. 2008. A Generalized Approach to Construct Benchmark Problems for Dynamic Optimization. In *Simulated Evolution and Learning*, Xiaodong Li, Michael Kirley, Mengjie Zhang, David Green, Vic Ciesielski, Hussein Abbass, Zbigniew Michalewicz, Tim Hendtlass, Kalyanmoy Deb, Kay Chen Tan, Jürgen Branke, and Yuhui Shi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 391–400.

[10] Michalis Mavrovouniotis, Shengxiang Yang, and Xin Yao. 2012. A Benchmark Generator for Dynamic Permutation-Encoded Problems. In *Parallel Problem Solving from Nature - PPSN XII*, Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 508–517.

[11] Trung Thanh Nguyen. 2011. *Continuous dynamic optimisation using evolutionary algorithms*. Ph.D. Dissertation. University of Birmingham.

[12] Philipp Rohlfshagen and Xin Yao. 2008. Attributes of Dynamic Combinatorial Optimisation. In *Simulated Evolution and Learning*, Xiaodong Li, Michael Kirley, Mengjie Zhang, David Green, Vic Ciesielski, Hussein Abbass, Zbigniew Michalewicz, Tim Hendtlass, Kalyanmoy Deb, Kay Chen Tan, Jürgen Branke, and Yuhui Shi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 442–451.

[13] Eric D. Taillard. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research* 64, 2 (1993), 278 – 285. https://doi.org/10.1016/0377-2217(93)90182-M Project Management anf Scheduling.

[14] Shengxiang Yang. 2003. Non-stationary problem optimization using the primal-dual genetic algorithm. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, Vol. 3. 2246–2253 Vol.3. https://doi.org/10.1109/CEC.2003.1299951

[15] Shengxiang Yang. 2015. Evolutionary Computation for Dynamic Optimization Problems. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15)*. ACM, New York, NY, USA, 629–649. https://doi.org/10.1145/2739482.2756589

[16] Abdunnaser Younes, Paul Calamai, and Otman Basir. 2005. Generalized Benchmark Generation for Dynamic Combinatorial Problems. In *Proceedings of the 7th Annual Workshop on Genetic and Evolutionary Computation (GECCO '05)*. ACM, New York, NY, USA, 25–31. https://doi.org/10.1145/1102256.1102262