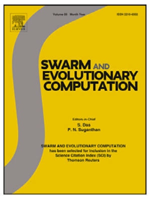
Swarm and Evolutionary Computation 70 (2022) 101044

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo



On the impact of linkage learning, gene-pool optimal mixing, and non-redundant encoding on permutation optimization

Arthur Guijt[a,b,*], Ngoc Hoang Luong[b,c], Peter A.N. Bosman[a,b], Mathijs de Weerdt[a]

[a] Delft University of Technology, Delft, the Netherlands
[b] Centrum voor Wiskunde en Informatica, Amsterdam, the Netherlands
[c] University of Information Technology (UIT), VNU-HCM, Viet Nam

ARTICLE INFO

Keywords:
Genetic algorithms
Estimation-of-distribution algorithms
Permutation problems

ABSTRACT

Gene-pool Optimal Mixing Evolutionary Algorithms (GOMEAs) have been shown to achieve state-of-the-art results on various types of optimization problems with various types of problem variables. Recently, a GOMEA for permutation spaces was introduced by leveraging the random keys encoding, obtaining promising first results on permutation flow shop instances. A key cited strength of GOMEAs is linkage learning, i.e., the ability to determine and leverage, during optimization, key dependencies between problem variables. However, the added value of linkage learning was not tested in depth for permutation GOMEA. Here, we introduce a new version of permutation GOMEA, called qGOMEA, that works directly in permutation space, removing the redundancy of using random keys. We additionally consider various linkage information sources, including random noise, in both GOMEA variants, and compare performance with various classic genetic algorithms on a wider range of problems than considered before. We find that, although the benefits of linkage learning are clearly visible for various artificial benchmark problems, this is far less the case for various real-world inspired problems. Finally, we find that qGOMEA performs best, and is more applicable to a wider range of permutation problems.

1. Introduction

Permutation problems are among the most important real-world problems. For example, scheduling problems, such as Permutation Flowshop or Jobshop Scheduling, and routing problems, such as Traveling Salesperson, are permutation problems that can be used to model real-world optimization tasks. Many of these problems belong to the class of NP-hard problems, making them very hard to solve to (near-)optimality. Evolutionary algorithms (EAs) are known to have much potential when it comes to finding high-quality solutions to such problems [1]. However, it is known that in the presence of strong non-linear dependencies between problem variables, without a proper configuration of how variation combines solutions, EAs may suffer from poor scalability as they may disrupt key building blocks in the mixing process [2]. Significant scalability improvements have been made by *linkage learning*, i.e., the automatic accounting for dependencies between problem variables by deriving and exploiting them during optimization, for instance through the estimation of probability distributions as done in Estimation of Distribution Algorithms (EDAs) [3,4]. Of particular interest are the more recent Gene–pool Optimal Mixing Evolutionary Algorithms - GOMEAs [5], which have been shown to be able to outperform EDAs on various benchmark problems for problems with discrete Cartesian variables as well as problems with real-valued variables by learning linkage more directly through information theoretic measures and by exploiting this information through more extensive solution mixing in each generation [5–7].

A variant of GOMEA, known as permutation GOMEA or pGOMEA [8], has been proposed to address problems in permutation spaces by making use of a random-key encoding scheme that represents permutations of variables [9]. The reported results are promising, with pGOMEA outperforming a state-of-the-art permutation EDA, i.e., GM-EDA, and a Permutation Flowshop Scheduling solver, i.e., VNS4, on many problem instances [8,10]. However, an in-depth analysis of the impact of learning and exploiting linkage structures of GOMEA in the permutation domain was not performed. Moreover, although the random-key encoding facilitates a straightforward design of permutation GOMEA, this encoding is also highly redundant due to variable permutations being encoded in the real-valued space. Such redundancy may affect the ability to efficiently detect dependencies and may harm the overall performance as well.

We here propose a new approach to the design of permutation GOMEA that circumvents these issues by operating on permutations directly, while still retaining the concepts of optimal mixing and linkage learning. We call this new version qGOMEA, going to the next letter in

* Corresponding author at: Centrum voor Wiskunde en Informatica, Amsterdam, the Netherlands.
E-mail address: arthur.guijt@cwi.nl (A. Guijt).

https://doi.org/10.1016/j.swevo.2022.101044
Received 17 May 2021; Received in revised form 27 December 2021; Accepted 15 January 2022
Available online 17 January 2022

the alphabet. We do this specifically as this new variant is significantly different due to the encoding used and operators employed, and to avoid confusion between them. In problems where positioning is fuzzy, the original approach still has its strengths, and hence cannot be replaced in its entirety.

We furthermore study the extent to which linkage learning actually influences the performance of GOMEA in the permutation domain, and how this performance compares to more conventional methods. We thus compare qGOMEA and the original approach pGOMEA, against various variants of the simple Genetic Algorithm (SimpleGA) using different crossover operators. The experiments are performed on a wide range of benchmark problems, chosen for their varying properties commonly seen in many permutation problems. In addition to the Permutation Flowshop Scheduling Problem, a multi-machine scheduling problem as seen in [8], we consider various benchmark instances from the Quadratic Assignment Problem [11], and Order Acceptance and Scheduling [12–14]. Most notably, the Order Acceptance and Scheduling problem has commonly been solved using local search approaches, such as Tabu Search in [12]. The successful hybridization with a population, such as in the memetic algorithm ALNS [14], hints at the potential effectiveness of Evolutionary Algorithms for this problem. Furthermore, this problem is a hybrid problem, containing both Travelling Salesperson, single-machine scheduling and knapsack-like traits. Moreover, scalability experiments are also conducted on the inversions benchmark function.

We summarize our contributions as follows:

- We note that Permutation GOMEA may disrupt certain structure without linkage learning being able to account for it due to the usage of Random Keys.
- We propose a new variant of GOMEA for permutation spaces, qGOMEA, which operates directly in permutation space, to resolve the aforementioned issue.
- We compare qGOMEA against Permutation GOMEA and a simple genetic algorithm utilizing classical permutation crossover operators.
- We show that using linkage learning Permutation GOMEA and qGOMEA can more effectively recombine on the Inversion Variants benchmark functions. Much like the impact of linkage learning seen for binary search spaces.
- We find that qGOMEA is more widely applicable than its precursor Permutation GOMEA. The results most notably show significantly improved performance for the Quadratic Assignment Problem. It is likely the encoding used in the original permutation GOMEA caused issues for these problems.
- We find that the effect of linkage learning on practical problems is unclear and problem dependent, in contrast to the results reported on a benchmark function. The evaluated problems either lack structure, or contain structure that was not learned effectively using current approaches.
- We conclude that improvements in how linkage learning is performed and applied are required for more generalizable linkage learning approaches.

The remainder of this paper is organized as follows. In Section 2 we describe the EAs that we have considered in addition to qGOMEA; which is described in detail in Section 3 itself. In Section 4 we then describe the permutation problems used for benchmarking. Subsequently, in Section 5 we describe the setup of our experiments, followed by a presentation of obtained results in Section 6. We discuss limitations and potential future work in Section 7, which is followed by our final conclusions in Section 8.

2. Evolutionary algorithms for permutation problems

This section provides relevant background of the proposed approach, as well as the relevant background of other approaches evaluated. We first give an overview of ways to encode a permutation as a solution – direct encoding and random keys – which are used in these approaches in Section 2.1. Then, in Sections 2.2–2.3.2, we give an overview of the background and structure of the evaluated approaches.

2.1. Representations

A permutation problem consists of n variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ with $x_i \in \{1, 2, \ldots, n\}$ such that $x_i \neq x_j$ for all $i \neq j \in \{1, 2, \ldots n\}$. I.e., each solution contains each element in $\{1, 2, \ldots n\}$ exactly once. Standard Cartesian recombination operators such as crossover can violate the uniqueness constraint of this integer representation, and as such cannot be used on integer representations directly. For this reason, specialized recombination operators are used, or the solution is encoded alternatively.

A popular alternative encoding is the random keys encoding by Bean [9]. It encodes permutations using n real values $\boldsymbol{r} = (r_1, r_2, \ldots, r_n)$, typically with $r_i \in [0, 1]$ for each $i \in \{1, 2, \ldots, n\}$. The encoded permutation can be obtained by sorting, i.e., it is $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_n)$ such that $r_{\pi_i} \leq r_{\pi_j}$ for all $i, j \in \{1, 2, \ldots n\}, i < j$. Consequently, standard crossover operators can be used.

2.2. Classic evolutionary algorithms

In this paper, we refer to classic EAs as EAs that perform recombination on full-length solutions to create new full-length offspring solutions, after which all offspring solutions are evaluated so as to be able to perform selection. Here, for simplicity, we use the same evolutionary scheme so as to be able to isolate and study the impact of the recombination operators solely. In particular, we use a P + O scheme with tournament selection and a tournament size of 4 so that the number of copies of the best solution grows logistically over time.

For the integer representation, we consider the specialized permutation recombination operators of Order Crossover (OX) [15], Partially Mapped Crossover (PMX) [16], Cycle Crossover (CX) [17] and Edge Recombination (ER) [18]. All of these operators operate on the integer representation of a permutation, and all – apart from ER – cross over a continuous substring of the solution. The general difference between these operators is the means by which they repair or preserve the uniqueness constraint. The way by which they do so can be briefly stated as follows. Order Crossover considers the original order to be key, and thus ensures this order is preserved by adding the items outside the crossed over substring back in their original order. Partially Mapped Crossover reduces individual disruptions by performing a swap between the item and its original position. A cycle crossover copies over cycles – substrings which contain the same items – crossing over extra elements to preserve uniqueness. Finally, Edge Recombination considers undirected edges key and constructs a complete solution by following edges in the solutions to unvisited items, selecting randomly from the unvisited pool if no such edge exists. For more details, we refer the interested reader to the aforementioned papers. For the random-key representation, we consider only uniform crossover.

While these operators can be regarded as old, they are well-known and understood. Furthermore, they perform well on many problems, especially when the structure they utilize matches the problem. This provides an indicator to how much of a particular kind of structure is present. This is combined with the Interleaved Multi-start Scheme, providing high-quality online population sizing. Ensuring that the population size is adjusted appropriately for any given problem configuration is exceedingly important, and yet an often missing aspect degrading the performance of these operators. As such these configurations together form a useful baseline and reference point in addition the original Permutation GOMEA. This approach has shown excellent performance on Permutation Flowshop against other state-of-the-art approaches, and is to be discussed next.

2.3. Gene-pool Optimal Mixing Evolutionary Algorithms

Gene-pool Optimal Mixing Evolutionary Algorithms (GOMEAs) are a form of EAs that combine the notion of crossover as found in classic EAs, with that of estimating and re-sampling probability distributions to model potential dependencies between problem variables as found in Estimation-of-Distribution Algorithms (EDAs) [5]. Below, we first explain key concepts of GOMEA as it was originally introduced for discrete Cartesian search spaces, followed by its adaptation to Permutation GOMEA that uses the random-key representation [5,8]. In Section 3 we propose qGOMEA, a new version of Permutation GOMEA that uses the integer representation.

2.3.1. Key concepts

Solutions $\boldsymbol{x}$ are encoded with n variables $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$ in the original GOMEA [5]. The solution space is the Cartesian product $\boldsymbol{x} \in \times_{i=1}^{n} \mathbb{D}_i$, where $\mathbb{D}_i$ is the domain of variable i.

A linkage model, that models dependencies between problem variables, is built every generation. For this, the Family of Subsets (FOS) concept is used. A FOS $\mathcal{F}$ contains $|\mathcal{F}|$ linkage sets, in which each linkage set is a subset of all problem variable indices, i.e., $\mathcal{F} = \{\boldsymbol{F}^1, \boldsymbol{F}^2, \dots, \boldsymbol{F}^{|\mathcal{F}|}\}$, where $\boldsymbol{F}^i \subseteq \{1, 2, \dots, n\}$. Variables in the same linkage set are considered to be dependent to some degree.

The most commonly used type of FOS is the Linkage Tree (LT). Conceptually, an LT is obtained by starting from n singleton sets, each consisting of one variable index, and then iteratively merging two sets until only one set remains with all indices. All sets that have been created in the process are part of the linkage tree. While an LT can be learned by computing a linkage metric between blocks of variables, this is costly. Instead, the LT in GOMEA is learned over population individuals by computing a (dis)similarity matrix based on linkage and applying a Hierarchical Clustering algorithm known as Unweighted Pair Group Method with Arithmetic Mean (UPGMA) in $\mathcal{O}(n^2 P)$ time, where P is the population size [5].

A common variant of the Linkage Tree is the Random Tree. A Random Tree is constructed much like a Linkage Tree, but the (dis)similarity used to construct the tree consists of symmetric random noise. Providing a similarly hierarchical collection of subsets, but without the additional complexity of using a linkage metric.

There is no separate selection step in GOMEA as there is in classic EAs. Instead, each generation, each solution $\boldsymbol{p}$ in the current population is transformed into an offspring solution $\boldsymbol{o}$ using the Gene-pool Optimal Mixing (GOM) operator. Considering each linkage set $\boldsymbol{F}^i$ of $\mathcal{F}$ in a random order, GOM incrementally tries to improve $\boldsymbol{p}$ by replacing values of variables indicated in $\boldsymbol{F}^i$ with values copied from a random donor in the population at the corresponding positions. If the *partial* modification yields an improved (or equal) fitness value, it is accepted. Otherwise, the changed variables revert to their previous values. If a solution is not improved during GOM or if the best solution ever found is not changed after a certain number of generations, a procedure called forced improvement (FI) is performed [8]. Essentially, FI performs GOM again, but with the current best solution as the donor and halting as soon as any improvement is achieved.

2.3.2. pGOMEA

Permutation GOMEA (pGOMEA) [8] uses the random-key encoding. Assigning a random key to each object (e.g. an order in PFS, a city in TSP, ...) as opposed to positions; as a position in a permutation is commonly closer to a continuous value than an arbitrary index. Because any mixing of random-keys always results in an encoding of a permutation, the GOM operator of Cartesian GOMEA can be straightforwardly used. When mixing a block of random-keys, only the relative sub-permutation pertaining to those keys is maintained. In other words, depending on the scaling of the random keys, the actual encoded integer permutation may still be different as this is dependent on the values of other random keys outside the block. Acknowledging this and attempting to improve performance, pGOMEA performs random *re-scaling*, where blocks of random keys are randomly re-scaled during mixing, and *re-encoding*, where the population is re-encoded after each generation with a new set of random-key strings that still maintains the underlying permutations of population individuals. This approach was successfully applied to Permutation Flowshop, outperforming GM-EDA in [8] and VNS4 in [10].

While the random-key encoding used in Permutation GOMEA preserves ordering information within a subset of variables when copying over their values, this is not the case for the absolute position and relative offset. Both absolute position and their offset are dependent on the other random keys in the solution. This results in potential disruption of building blocks during recombination for problems in which such structure is more important than ordering alone. This is contrary to what GOMEA is trying to achieve through linkage learning.

It should be noted that this mismatch is inherent to the operator and encoding. The disruption that occurs when the aforementioned traits are important cannot be completely avoided through the selection of the right subset. As linkage learning operates by determining the right subsets, linkage learning will not be able to avoid disruption either. As this nullifies the benefits of linkage learning, an alternative is necessary.

By operating directly in permutation space we can more freely preserve linkage. However, encoding directly in permutation space is not without its issues.

First of all, in pGOMEA the uniqueness constraint is being preserved by the use of the random-key encoding. When operating in permutation space directly this becomes the responsibility of the operators used instead.

Furthermore, Permutation spaces are large; $O(n!)$ compared to $O(2^n)$ for binary Cartesian problems. The operators used should utilize the structure that permutations provide in order to operate efficiently within this domain.

An example of such structure would be that one order b always follows order a. If such patterns is an important aspect of a good solution, an operator should be able to encode this pattern efficiently and apply it in other situations.

E.g. an EA utilizing only PMX as a crossover operator will have trouble reproducing this pattern under translation, requiring a schema for each position.

In pGOMEA this pattern is encoded by using random keys in combination with re-encoding and re-scaling. In the new approach we will use a notion of locality, similar to that of local search approaches.

3. qGOMEA

In this section we introduce qGOMEA and its new operators, the *Differential Crossover* in Section 3.2 and *Reorder Crossover* in Section 3.3. These operators replace the copying operation within GOM, creating a variant of GOM preserving uniqueness constraints.

Much like how the copying operation in binary GOM acts like a population-informed multi-variable generalization of a single-variable bit-flip; these operators perform actions generalizing those of the block move and swap local search operators, allowing for utilization of common notions of locality for permutation problems. The remainder of this section describes the other differences from Permutation GOMEA, most notably the removal of operations working on Random Keys, and the linkage metric in Section 3.4. A pseudocode overview of qGOMEA can be found in Algorithm 1.

3.1. Notation

Before describing the actual operators themselves, we first describe some supplementary notation.

First of all, apart from numbering *positions* 1 through n, we define $\mathcal{A}$ to number *objects* – e.g., the cities in TSP, jobs in scheduling problems, ... – by letters: $\mathcal{A} = a, b, \dots, y, z, aa, ab, \dots$ in order to clearly differenti-

Algorithm 1 qGOMEA.

```
1: procedure StepqGOMEA(population)    ▷ Run a single generation of qGOMEA
2:     fos ← LearnFOS(population)
3:     Copy population to offspring
4:     for all $o \in$ offspring do
5:         improved ← GOM(o, fos, population)
6:         if not improved OR NIS reached then
7:             FI(o, fos, population)
8:     return offspring
9: procedure LearnFOS(population)
10:    D ← DetermineD(population)    ▷ Obtain dissimilarity matrix
11:    lt ← UPGMA(D)    ▷ Learn tree through hierarchical clustering
12:    fos ← toFOS(lt)
13:    fos ← Filter(fos)    ▷ Filter preserving elements of size $< 5/6n$
14:    return fos
15: procedure GOM(solution, fos, population)
16:    improved ← False
17:    for all $s \in$ fos do
18:        $d$ ← RandomSample(population)
19:        for all $op \in$ {Differential Crossover, Reorder Crossover} do
20:            $s'$ ← RecombineEvaluate(s, d, fos, $op$)
21:            if $s'$ is better in fitness than $s$ then
22:                improved ← True
23:            if $s'$ is better or equal in fitness to $s$ then
24:                Replace $s$ with $s'$
25: procedure FI(solution, fos, population)
26:    $d$ ← Elitist(population)
27:    for all $s \in$ fos do
28:        for all $op \in$ {Differential Crossover, Reorder Crossover} do
29:            $s'$ ← RecombineEvaluate(s, d, fos, $op$)
30:            if $s'$ is better in fitness than $s$ then
31:                Replace $s$ with $s'$
32:                return
```

ate between positions and objects. Furthermore, much like we use k to indicate the kth position, we use $\mathcal{A}_k$ to indicate the kth object.

In general, we refer to solutions in a Permutation space $\mathbb{P}_n$, as a one-to-one assignment of *objects* to *positions*. For example in the case of $P \in \mathbb{P}_5$:

$$P = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{bmatrix} a & d & b & c & e \end{bmatrix} \end{array}$$

Such a solution has a corresponding assignment P^{-1} of *positions* to *objects* as well in the corresponding inverse Permutation space $\mathbb{P}_n^{-1}$. For example:

$$P^{-1} = \begin{array}{c} \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{bmatrix} 1 & 3 & 4 & 2 & 5 \end{bmatrix} \end{array}$$

An alternative perspective on permutations is that they define a discrete Cartesian space of size n, with each dimension having n possible values, and the constraint that the value for each dimension must be different. Given a permutation $P \in \mathbb{P}_n$:

$$P_x \neq P_y \; \forall \; x, y \in [1, \dots, n], \; x \neq y \tag{1}$$

This holds similarly for the inverse $P^{-1} \in \mathbb{P}_n^{-1}$ as well. As before, in this space we index using the alphabet $\mathcal{A}$ instead to avoid overlap between the indices.

Rather than performing a crossover by position, it may be more meaningful to perform a crossover on the differences in position instead. For example when a permutation encodes a sequence, as is the case with the Traveling Salesperson and many Machine Scheduling problems.

Additionally, the constraint above can be rewritten to make use of differences instead. A crossover on differences will therefore preserve the constraints as well as a standard crossover would.

$$P_x^{-1} - P_y^{-1} \neq 0 \; \forall \; x, y \in [\mathcal{A}_1, \dots, \mathcal{A}_n], \; x \neq y \tag{2}$$

3.2. Differential crossover

Given the commonality of problems in which a permutation is used as a sequence, the first crossover operator in qGOMEA – Differential Crossover – is a hybrid between a crossover for absolute and relative positioning.

The goal of this operator is to obtain a population-informed generalization of the block-move operator, commonly used in local search approaches to permutation optimization. It generalizes this operator by using a subset rather than a continuous range / block of the permutation. Furthermore, it does so preserving the offsets - moving only the anchor to the position of the donor.

All mixing operators in GOMEA are provided a linkage subset from the FOS, much like most genetic algorithms select a substring from the two parents to interchange.

This operator assumes relative positioning, rather than the absolute values themselves, to be key within the subset. Relative positioning requires a reference point – the anchor. We pick the anchor from the linkage subset. This operator moves the anchor to the position of the donor, preserving the offsets of the original solution within the subset itself. Pseudocode for this operator can be found in Algorithm 2.

Algorithm 2 Differential Crossover.

```
1: procedure DifferentialCrossover(P, Q, $\mathcal{S}$)
2:     % Determine virtual donor
3:     $A$ ← RandomSample($\mathcal{S}$)    ▷ Pick Anchor index
4:     for all $i \in \mathcal{S}$ do    ▷ Determine virtual donor
5:         $D^{-1} \leftarrow Q_A^{-1} + (P_A^{-1} - P_i^{-1})$
6:     $O \leftarrow \max\{0, \max_{j \in \mathcal{S}}\{D_j^{-1}\} - L + 1\}$    ▷ Determine overflow
7:     $U \leftarrow \min\{0, \min_{j \in \mathcal{S}}\{D_j^{-1}\}\}$    ▷ Determine underflow
8:     for all $i \in \mathcal{S}$ do    ▷ Repair by shifting
9:         $D_i'^{-1} \leftarrow D_i^{-1} - \max\{0, \max_{j \in \mathcal{S}}\{D_j^{-1}\}\}$
10:    % Recombine (OX)
11:    for all $i \in \mathcal{S}$ do    ▷ Place in donor items
12:        $R_{D_i^{-1}} \leftarrow i$
13:        Mark position $D_i^{-1}$ as taken.
14:    $j \leftarrow 0$
15:    for all $i \in P$ do    ▷ For each item as ordered in P
16:        if $i \in \mathcal{S}$ then    ▷ Skip over items already in R
17:            continue
18:        while $j$ is taken do    ▷ Skip over positions already in R
19:            $j \leftarrow j + 1$
20:        $R_j \leftarrow P_i$
21:        $j \leftarrow j + 1$
22:    return R
```

Given a linkage subset $\mathcal{S} \in \mathcal{F}$ from the FOS consisting of objects, and solutions P and Q. First, the anchor $A \in \mathcal{S}$ is chosen randomly within the subset. Using A as the reference point a new virtual donor D is constructed, such that D^{-1} satisfies:

$$D_i^{-1} = Q_A^{-1} + (P_i^{-1} - P_A^{-1}) \; \forall \; i \in \mathcal{S} \tag{3}$$

It is possible that the resulting donor is invalid due to the position ending up outside the domain of $\mathbb{P}_n^{-1} : [1, n]$. This would lead to an invalid permutation. We choose to resolve this issue by shifting back by the overflow O or forward by the underflow U.

$$O = \max\{0, \max_{j\in\mathcal{S}}\{D_j^{-1}\} - n + 1\}$$

$$U = \min\{0, \min_{j\in\mathcal{S}}\{D_j^{-1}\}\}$$

And then construct a partial solution D' such that:

$$D_i'^{-1} = D_i^{-1} - O - U \;\forall\, i \in \mathcal{S}$$

This virtual donor D, or D' if an error occurred, is then used in a crossover on P, copying the subset $\mathcal{S}$, performing repairs similarly to the Order Crossover operator. This results in a new candidate solution R for evaluation.

For example, given

$$P = \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ \hline a & b & d & c & e \end{array} \qquad P^{-1} = \begin{array}{ccccc} a & b & c & d & e \\ \hline 1 & 2 & 4 & 3 & 5 \end{array}$$

$$Q = \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ \hline e & d & c & b & a \end{array} \qquad Q^{-1} = \begin{array}{ccccc} a & b & c & d & e \\ \hline 5 & 4 & 3 & 2 & 1 \end{array}$$

$$\mathcal{S} = \{d, e\}$$

We first select an Anchor $A \in \mathcal{S}$

$$A = d$$

As such $P_A^{-1} = P_d^{-1} = 3$ and $Q_A^{-1} = Q_d^{-1} = 2$.

By Eq. (3) this results the following facts

$$D_d^{-1} = 2 + (3 - 3) = 2 \qquad D_e^{-1} = 2 + (5 - 3) = 4$$

The resulting virtual donor D is therefore

$$D = \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ \hline \_ & d & \_ & e & \_ \end{array} \qquad D^{-1} = \begin{array}{ccccc} a & b & c & d & e \\ \hline \_ & \_ & \_ & 2 & 4 \end{array}$$

This resulting donor is then used construct a complete solution R. As $\mathcal{S} = \{d, e\}$, the remaining elements are $\bar{\mathcal{S}} = \{a, b, c\}$. Which are ordered `[a, b, c]` in P. The holes in D are filled in from left-to-right in this same order. Yielding R:

$$R = \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ \hline a & d & b & e & c \end{array} \qquad R^{-1} = \begin{array}{ccccc} a & b & c & d & e \\ \hline 1 & 3 & 5 & 2 & 4 \end{array}$$

The donor in this case was valid, however, the resulting donor can be invalid. Example of an invalid donor D_{err}:

$$D_{\text{err}} = \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ \hline \_ & \_ & \_ & \_ & d & e \end{array} \qquad D_{\text{err}}^{-1} = \begin{array}{ccccc} a & b & c & d & e \\ \hline \_ & \_ & \_ & 5 & 6 \end{array}$$

Resolving this by shifting would result in D'_{err} becoming:

$$D'_{\text{err}} = \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ \hline \_ & \_ & \_ & d & e \end{array} \qquad D'^{-1}_{\text{err}} = \begin{array}{ccccc} a & b & c & d & e \\ \hline \_ & \_ & \_ & 4 & 5 \end{array}$$

3.3. Reorder crossover

Yet another perspective on Permutations lies at the basis of the second operator used in qGOMEA. A permutation may be seen as a transitive ordering of elements as well. For example, instead of the objective being dependent on the exact position or the exact difference between two dimensions, it may be dependent on the ordering implied by the positions. Reusing P and Q from the previous example, an alternative way to denote this permutation would be:

$$P = a < b < d < c < e$$

$$Q = e < d < c < b < a$$

The idea of this operator is to transfer over the ordering of a subset, restricted to the subset. For example, when transferring the ordering for a pair, this will either lead to doing nothing when the ordering is already the same, or performing a swap otherwise.

More generally, the operator reorders the indices in a solution P, restricted to the linkage subset $\mathcal{S}$, such that the ordering within $\mathcal{S}$ is equal to that of Q. In particular, this operator is identical to UOX operator described in [19,20] with the mask originating from the Family of Subsets instead. Pseudocode for this operator can be found in Algorithm 3.

Algorithm 3 Reorder Crossover.

```
1:  procedure ReorderCrossover(P, Q, $\mathcal{S}$)
2:      % Determine mapping
3:      $L_P \leftarrow [P_i^{-1}$ for $i \in \mathcal{S}]$
4:      $L_Q \leftarrow [Q_i^{-1}$ for $i \in \mathcal{S}]$
5:      $L_P^{\text{sorted}} \leftarrow \text{Sort}(L_P)$
6:      $L_Q^{\text{sorted}} \leftarrow \text{Sort}(L_Q)$
7:      $\mathcal{M} \leftarrow \{q \rightarrow p$ for $p, q \in \text{zip}(L_P^{\text{sorted}}, L_Q^{\text{sorted}})\}$ ▷ Construct mapping
        by corresponding indices
8:      % Use mapping
9:      Copy P to R.
10:     for all $i \in \mathcal{S}$ do
11:         $R_i^{-1} \leftarrow M[Q_i^{-1}]$
12:     return R
```

Formally, given the recipient P, the donor Q and the linkage subset $\mathcal{S} \subseteq \mathcal{A} \in \mathcal{F}$, with $\bar{\mathcal{S}} = \mathcal{A} - \mathcal{S}$. The resultant is the solution R such that:

$$R_i^{-1} = P_i^{-1} \qquad \text{for } i \in \bar{\mathcal{S}} \tag{4}$$

$$R_i^{-1} > R_j^{-1} \qquad \text{if } Q_i^{-1} > Q_j^{-1} \text{ for } i, j \in \mathcal{S}, i \neq j \tag{5}$$

This resultant R can be created as follows. Construct two lists

$$L_P = [P_i^{-1} \text{ for } i \in \mathcal{S}]$$

$$L_Q = [Q_i^{-1} \text{ for } i \in \mathcal{S}]$$

Sort both lists yielding, L_P^{sorted} and L_Q^{sorted}. Construct a mapping $\mathcal{M} : L_Q^{\text{sorted}} \rightarrow L_P^{\text{sorted}}$ which maps elements by index from Q to P (i.e., such that the first item of L_Q^{sorted} in is mapped to the first item of L_P^{sorted}).

Now initialize R to be a copy of P, and perform

$$R_i^{-1} = \mathcal{M}(Q_i^{-1}) \qquad \text{for } i \in \mathcal{S}$$

Assuming the P and Q stated previously, and $\mathcal{S} = \{a, b, c\}$:

$$P = [a \;\; b \;\; d \;\; c \;\; e] \qquad L_P = [1, 2, 4] \qquad L_P^{\text{sorted}} = [1, 2, 4]$$

$$Q = [e \;\; d \;\; c \;\; b \;\; a] \qquad L_Q = [5, 4, 3] \qquad L_Q^{\text{sorted}} = [3, 4, 5]$$

The mapping $\mathcal{M}$ is therefore $\mathcal{M} : \{3 \rightarrow 1, 4 \rightarrow 2, 5 \rightarrow 4\}$. Additionally, as R is initially a copy of P:

$$R'^{-1} = P^{-1} = \begin{array}{ccccc} a & b & c & d & e \\ \hline 1 & 2 & 4 & 3 & 5 \end{array} \qquad Q^{-1} = \begin{array}{ccccc} a & b & c & d & e \\ \hline 5 & 4 & 3 & 2 & 1 \end{array}$$

Finally, R is constructed by apply the mapping $\mathcal{M}$ for all elements in $\mathcal{S}$:

$$R_a^{-1} = \mathcal{M}(Q_a^{-1}) = \mathcal{M}(5) = 4$$

$R_b^{-1} = \mathcal{M}(Q_b^{-1}) = \mathcal{M}(4) = 2$

$R_c^{-1} = \mathcal{M}(Q_c^{-1}) = \mathcal{M}(3) = 1$

Resulting in the solution R:

$$R = \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ \hline c & b & d & a & e \end{array} \qquad R^{-1} = \begin{array}{ccccc} a & b & c & d & e \\ \hline 4 & 2 & 1 & 3 & 5 \end{array}$$

3.4. Configuration

The new operators allow qGOMEA to operate on permutations directly instead of using the random-key encoding.

The new operators are not the only differences in qGOMEA. In general, in a GOMEA the mixing operators use subsets of elements from a Family of Subsets (see Section 2.3.1). In order to build a linkage tree as used in Permutation GOMEA, a measure for linkage (or the dependency strength) needs to be defined. The pairwise linkage was derived from multiplying the inverse entropy of ordering with the average distance between the two items.

However, in many permutation problems the fitness of an item is dependent on the one preceding. An item that often precedes another hence often forms a 'block'. Conversely, the order does not necessarily matter, for example in a symmetric problem, and may even be misleading. Items vying for the same spot, or one close to one another, may change ordering due to repairs, causing the inverse entropy to be notably lower for nearby items. Resulting in known linkage in the structure of the problems, including Permutation Flowshop, the original problem Permutation GOMEA was tested on, to be ignored. qGOMEA utilizes only distance as its – dissimilarity – linkage metric. Given the population $\mathcal{P}$ consisting of solutions $pi \in \mathbb{P}^{-1}$ where objects are mapped to positions.

$$D(i,j) = \sum_{\pi \in \text{population}} \left|\pi_i - \pi_j\right| \tag{6}$$

Due to the aforementioned removal of random keys and its mutation operators, qGOMEA has less diversification operators than the original Permutation GOMEA. In order to preserve the diversity that we have, we disable Forced Improvement. FI reduces diversity considerably by mixing with the best solution found so far.

Furthermore, mixing a larger subset into a solution has a larger likelihood of turning a solution into a copy of the other, effectively reducing diversity. This effect is further exacerbated by the repairs performed to preserve the uniqueness constraint. In order to prevent this, we prune the Linkage Tree, keeping only the subsets of size $< 5/6n$.

As we do not expect the performance of the approach to be sensitive to the exact value given that the value is large enough; a tree mostly contains smaller nodes. As such we did not extensively tune this parameter. We tested values $1/2n$, $3/4n$, $5/6n$ and $11/12n$ on the inversion problem with $n = 100$ to find the largest value where premature convergence does not occur.

4. Permutation problems

In this section we describe the problems used in order to assess the performance of the aforementioned approaches. The first set of problems are chosen to both assess the performance in an idealized case, improving our understanding of our approaches and the linkage learning aspects in a controlled environment. The second set focuses on problems that are more representative of real-world problems, such as scheduling problems. The goal of this set is to assess the performance and applicability of the proposed approach on function landscapes that are more likely to occur in real-world situations.

The first set of problems consist of the Number of Inversions function and variants. These functions will be used to determine the scalability of the approaches as the problem becomes larger and more difficult. This will also allow us to evaluate how well linkage learning works, as scalability is the key improvement linkage learning provides. Furthermore, due to the sparse nature of the functions, these functions also benchmark how well the approaches recombine information from different solutions to navigate plateaus and local optima.

As these benchmark functions are not necessarily representative of real-world problems, we additionally assess the performance on a variety of more realistic benchmark problems. This allows us to evaluate how capable of exploiting linkage the currently evaluated approaches are in a more practical setting. Each problem is representative of a different class such permutation problems: Permutation Flowshop is related to multi-machine scheduling, Order Acceptance and Scheduling with Sequence Dependent Setup Times is a (complex) single-machine scheduling problem, and the Quadratic Assignment Problem as representatives is an assignment problem.

4.1. Inversion problems

In this subsection we describe a family of functions whose objective relates to the difference in ordering of the given permutation against the reference permutation $[1, \dots, n]$.

The first function in this family – Number of Inversions – sums over the product of all pairs in the permutation. This results in a function that is easily solvable by hill climbing; after each swap there is immediate feedback of whether this was a step in the right direction or not. The other two inversion problem functions described in this section aim to be more difficult by leaving out information by only awarding the score in a fraction of the cases – creating a plateau that will have to be navigated by the approaches. Therefore, these functions will be harder to navigate and will require recombination to find a global optimum, unlike the first function.

Number of inversions The Number of Inversions function as defined in the literature counts the number of inversions, i.e., the number of incorrectly sorted pairs. The function according to this definition should be minimized. All other functions defined below are maximization functions. For consistency, we define the Number of Inversions function to count the number of correctly sorted pairs instead: this definition is equivalent, except that this function should be maximized.

$$f_1(\pi) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \begin{cases} 1 & \text{if } \pi_j > \pi_i \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Number of sequential inversions This first variant only evaluates sequential pairs, rather than the product of pairs, i.e., it counts the number of cases in which two sequential items are ordered correctly.

$$f_2(\pi) = \sum_{i=1}^{n-1} \begin{cases} 1 & \text{if } \pi_{i+1} > \pi_i \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The primary difference between the original and this variant lies in the landscape of the improving moves. Take for example the solution $\pi =$ `[3, 4, 1, 2]`, for which $f_1(\pi) = 2$ and $f_2(\pi) = 2$. While improving swaps exist for the original objective function, no swap will yield a higher objective value for f_2 pairs. In fact, only the optimal solution $\pi =$ `[1, 2, 3, 4]` has a better objective value with $f_2(\pi) = 3$.

Number of sequential pairs The second and last variant are similar to the previous problem; this function also sums over sequential pairs, not the product. But rather than being well-ordered, it requires the numbers in the pair to be sequential numbers. This property is more strict and less likely to appear in a random solution than in a well-ordered one. Yet it may be easier: all pairs that cause the objective value to increase, appear in the optimal solution as well.

$$f_3(\pi) = \sum_{i=1}^{n-1} \begin{cases} 1 & \text{if } \pi_{i+1} = \pi_i + 1 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

4.2. Permutation Flowshop

The first more realistic problem is the Permutation Flowshop problem [21]. In this problem there are m machines on which n jobs need to be performed. Each job has m tasks which need to be performed on a corresponding machine each – i.e., the k-th task of a job, is performed on the k-rd machine – each costing a particular amount of time. A task can only be performed once prior tasks of the same job have been completed and the corresponding machine is available.

Given a permutation π of length n, job i, machine j, and corresponding processing time $p(i,j)$, the completion time $c(i,j)$ can be defined as:

$$c(\pi_1, 1) = p(\pi_1, 1) \tag{10}$$

$$c(\pi_1, j) = c(\pi_1, j-1) + p(\pi_1, j) \tag{11}$$

$$c(\pi_i, 1) = c(\pi_{i-1}, 1) + p(\pi_i, j) \tag{12}$$

$$c(\pi_i, j) = \max\{c(\pi_{i-1}, j), c(\pi_i, j-1)\} + p(\pi_i, j) \tag{13}$$

The resulting objective is to *minimize* either the maximum time spent – the Max Flow time – or the sum over each job – the Total Flow Time – respectively.

$$f_{\max}(\pi) = c(\pi_n, m) \tag{14}$$

$$f_{\text{total}}(\pi) = \sum_{i=1}^{n} c(\pi_i, m) \tag{15}$$

4.3. Quadratic Assignment

The Quadratic Assignment problem is a combinatorial optimization problem in which facilities with flows need to be assigned to locations with distances between them. The goal then is to minimize the product of distances with flows.

Unlike the previously listed problems, the interpretation of the underlying permutation is no longer that of a sequence, but rather that of a one-to-one assignment. For this reason we expect this problem to have notably different performance characteristics.

An instance is parameterized by two $n \times n$ matrices A and B. Given a permutation π the function to be *minimized* is

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} A(i,j) \cdot B(\pi_i, \pi_j) \tag{16}$$

4.4. Order acceptance and scheduling with sequence dependent setup times

Many real world problems are more complex than the Permutation Flowshop problem. Many real world problems are hybrids of other – simpler – problems. In this section we cover the hybrid problem by Oğuz et al. [13] incorporates aspects of Traveling Salesperson, Scheduling Problems and Knapsack in one problem. In short, one could interpret this problem as a knapsack problem where weight is replaced by time, with the allocation of time involving a scheduling problem with TSP-like traveling time between jobs.

More specifically, following Oğuz et al. [13] a problem instance is parameterized for a given permutation π, and n orders $i \in \{1, n\}, j \in \{0, n\}$ – where 0 is the initial order, using a collection of n release times r_i, processing times p_i, due dates d_i, deadline $\bar{d}_i$, revenue e_i, penalty weight w_i and $n(n+1)$ sequential setup times $s_{j,i}$.

In the original problem specification an order can go past their deadline and cause a solution to be invalid. This resulted in the solutions being sequences, where each order appeared at most once. This is different from permutations, where each order should appear *exactly* once. To ensure every permutation encodes a valid solution, we use the approach by Chaurasia and Singh [22]. In their work they propose to ignore such orders entirely during evaluation. This results in the introduction of prev in the following formulas; it keeps track of the previous order in the solution – ignoring any dropped orders – such that the sequence dependent setup time can be calculated properly.

A solution can be evaluated as follows.

$$\text{prev}(\pi_1) = 0 \tag{17}$$

$$\text{prev}(\pi_i) = \begin{cases} \pi_{i-1} & \text{if } c\prime_{\pi_{i-1}} < \bar{d}_{\pi_{i-1}} \\ \text{prev}(\pi_{i-1}) & \text{otherwise} \end{cases} \tag{18}$$

$$c'(\pi_1) = r(\pi_1) + s_{0,\pi_1} + p_{\pi_1} \tag{19}$$

$$c'(\pi_i) = \max\{c(\pi_{i-1}), r_{\pi_i}\} + s_{\text{prev}(\pi_1),\pi_1} + p_{\pi_1} \tag{20}$$

$$c(\pi_i) = \begin{cases} c'(\pi_i) & \text{if } c'(\pi_i) < \bar{d}_{\pi_i} \\ c(\pi_{i-1}) & \text{otherwise} \end{cases} \tag{21}$$

Every order has a corresponding profit. The full revenue is awarded if an order was completed before the due date. If an order was completed after the due date, but before the deadline, a penalty is subtracted from the revenue for each time unit after the due date. Orders completed at or after the deadline are ignored and do not incur any profit.

The objective of this problem is to maximize the sum of these profits.

$$f(\pi_i) = \begin{cases} e_{\pi_i} & \text{if } c'(\pi_i) < d_{\pi_i} \\ e_{\pi_i} - w_{\pi_i} \cdot (c'(\pi_i) - d_{\pi_i}) & \text{if } c'(\pi_i) < \bar{d}_{\pi_i} \\ 0 & \text{otherwise: } c'(\pi_i) \geq \bar{d}_{\pi_i} \end{cases} \tag{22}$$

$$f_{\text{total}}(\pi) = \sum_{i=1}^{n} f(\pi_i) \tag{23}$$

5. Experimental setup

All experiments are performed on an Intel Core i7-8750H CPU @ 2.20GHz, with 16 GB of RAM, running Windows, unless noted otherwise. The approaches are implemented in Julia,[1] and evaluated in Julia 1.3, using a single thread. Approaches are as follows, and summarized in Table 1.

In addition to the GOMEAs – pGOMEA and qGOMEA – with both Linkage Tree and Random Tree (implemented as a linkage tree based on random linkage) models, we also evaluate the performance of standard permutation crossover operators – CX[17], PMX[16], OX[15], ER[18] – and uniform crossover on Random Keys [9] in a simple-GA setup. In this setup each generation individuals are randomly paired up, with each mating pair producing two offspring. Both the parents and their offspring compete in a tournament of size 4, selecting the two best solutions. ER was dropped from tables due to extremely poor performance.

All approaches use the same Interleaved Multi-start Scheme (IMS) as the GOMEAs. This scheme operates on various populations of increasing size in an interleaved fashion. Starting at a particular base population size pop_{base} a generational step is being performed by said population. After running pop_{f} generations of a size, the next rank up runs once. Note that this pattern recurses. If there exists no next rank up, one is created with double the population size. While populations that have converged or no longer provide any improvement are pruned.

This removes the need to configure the population size without a significant loss in performance. For the GOMEAs the parameters are set to $pop_{\text{base}} = 4$ and $pop_{\text{f}} = 4$. As the standard GAs perform notably fewer evaluations per generation ($O(n)$ compared to $O(n^2)$), these values are set to $pop_{\text{base}} = 16$ and $pop_{\text{f}} = 8$ to avoid excessive population growth and corresponding memory usage.

[1] Source code can be found at https://github.com/8uurg/GeneticPermutationBenchmark

Table 1
Table summarizing the listing of approaches evaluated in this work.

Approach	Population Sizing		Operators
	pop_{base}	pop_{f}	
Random Key SimpleGA	16	8	UX
Integer Permutation SimpleGA	16	8	CX [17] / PMX [16] / OX [15]
Permutation GOMEA [8]	4	4	Copy (LT/RT)
qGOMEA	4	4	Reorder & Differential (LT/RT)

Apart from population size, qGOMEA and the SimpleGAs are parameterless, requiring no further tuning of parameters. Permutation GOMEA contains an additional parameter, the probability for the re-scaling operator. The impact of this parameter is dependent on the problem and instance. This would therefore require per-problem tuning. As none of the aforementioned problems employ per instance tuning ahead of time, doing so would incur an unfair advantage. As such we utilize the default, $p = .1$, which we have validated to work well overall.

For each problem, all results are reported as the median of the gap to the optimum as a percentage. In case no optimum is known, bounds on this solution are used instead, as specified in respective problem section. Results in bold either correspond to the approach with the best mean gap, or are not statistically significantly different according to the Mann-Whitney U-test from this approach with $p < \frac{0.05}{8}$ – with the Bonferroni correction applied. The background color of each cell is colored proportional to the value, compared against the other approaches for the same instance. The best performer on an instance is therefore colored with a white background, whereas the worst performer has a dark gray background.

For each of the permutation problems in Section 4 an experiment is performed. These experiments are described in the following sections.

5.1. Inversion benchmark

The experiments for the Inversion variants benchmark functions are defined to be a scalability experiment, in which we measure the number of evaluations that are required to reach the optimum in 90% of the runs.

A total of 10 runs are performed for each approach/function/n combination, with a time limit of 100. If the approach had no successful runs for n on the same function, we skip evaluating larger n. Possible values for n are $10, 15, 20, 25, 50, 100, 200, 400$ and 800.

For the inversion benchmark function the optimum is known ahead of time to be the ascending ordering: $(1, 2, \dots, n)$. As this ordering is easier to find due to initialization biases and programmatic errors, we randomly rename the items but not positions in half of the cases to remove any potential bias to this solution.

5.2. Permutation Flowshop

We perform an experiment similar to the one performed for Permutation GOMEA in [8]. We use the same configuration as [23], but utilize the max-flow objective instead of the total flow. For this experiment Taillard's instances [21] are used, using the bounds on his website [24]. Every approach is given a computational budget of evaluations according to n and m, as stated in Table 2. A total of 20 runs is performed for each approach/instance combination. Because of the higher computational workload, we performed this set of experiments on a server with 64 cores (AMD Opteron Processor 6386 SE, 2.8 GHz). The experiment was performed using Julia 1.4.2 using a single thread.

5.3. Quadratic Assignment Problem

The QAP instances originate from QAPLIB [11] using the upper bounds on the optimal solutions reported on the website [25]. For the

Table 2
Computational budget of evaluations, dependent on n and m, from Ceberio et al. [23], for permutation Flowshop.

J x M	# evaluations	J x M	# evaluations
20 x 5	182 224 100	100 x 5	235 879 800
20 x 10	224 784 800	100 x 10	266 211 000
20 x 20	256 896 400	100 x 20	283 040 000
50 x 5	220 712 150	200 x 10	272 515 500
50 x 10	256 208 100	200 x 20	287 728 850
50 x 20	275 954 150	500 x 20	260 316 750

Quadratic Assignment Problem the time budget for each run is 10 minutes, independent of dimensionality. Each approach/instance configuration has a total of 12 runs – but the first run is dropped.

The approaches were evaluated on the same hardware as Permutation Flowshop; on a server with 64 cores (AMD Opteron Processor 6386 SE, 2.8 GHz) running Julia 1.4.2.

5.4. Order Acceptance and Scheduling

We use the instances by Oğuz et al. [13], with bounds on the optimal solutions for these instances provided by Silva et al. [26]. These instances are parameterized by τ, R and n; with $\tau \in [0.1, 0.3, 0.5, 0.7, 0.9]$ being the variability in release times, $R \in [0.1, 0.3, 0.5, 0.7, 0.9]$ being the variability in the availability window (i.e., $\bar{d_i} - r_i$) and $n \in [10, 15, 20, 25, 50, 100]$ being the number of orders. Each run has a time budget of $n^2 + 100$ ms – e.g., 100.1 s for $n = 100$. Each approach/instance combination has a total of 11 runs.

6. Results

6.1. Inversion Variants

The scalability of the approaches on each of these three benchmark functions can be found in Fig. 1. This graph shows the 90th percentile of the number of evaluations required to find the optimum (lower is better) given a problem of certain size. Both axes are plotted logarithmically, (low) polynomial scalability will result in a straight line (increasing with size of the problem), whereas exponential growth will cause the graph to curve upwards.

In general, the SimpleGA using Edge Recombination crossover performs badly. This is likely due to the symmetry assumption used in the implementation, while none of the benchmark functions are symmetric. In fact, the reverse of the optimum is for every inversion variant the worst solution possible. All other approaches scale well on the inversion benchmark, whereas the variants – Sequential Inversion and Sequential Pairs – appear to be more difficult in general for all approaches evaluated.

In particular, qGOMEA outperforms on the Sequential Inversion function independent of the linkage structure used. Furthermore, unlike the other approaches qGOMEA exhibits polynomial scalability. As pGOMEA does not display polynomial scalability and the improvement is independent of linkage structure used, we can conclude that qGOMEA's new operators provide a considerable performance and scalability improvement on the Sequential Inversion benchmark function.

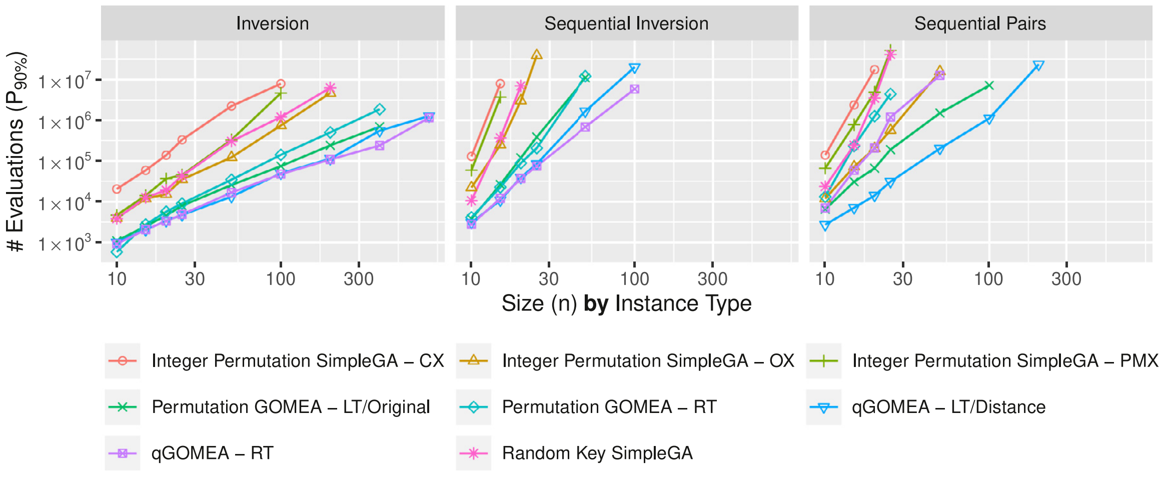


Fig. 1. Scalability for the inversion variants benchmark functions.

For Sequential Pairs a clear benefit for approaches utilizing Linkage Learning appears: both Permutation GOMEA and qGOMEA have notably improved performance when using a Linkage Tree as opposed to a random tree. This is caused by the blocks being used by both qGOMEA and Permutation GOMEA in performing Differential Crossover and Random Rescaling respectively. In combination with the Linkage Tree containing the right building blocks these operations exemplify the prowess of linkage learning: recombining smaller correctly ordered subsequences into larger correctly ordered subsequences.

The lack of this occurring for the other two benchmarks indicates that the current configuration is far from perfect however, and the efficacy of linkage learning and its current metric is strongly dependent on the problem itself.

6.2. Permutation Flowshop

The results for the Permutation Flowshop benchmark are listed in Table 3.

The GOMEA family of approaches outperform all SimpleGAs on the Permutation Flowshop problem instances. We note that linkage learning provides only a minor benefit, with the Linkage Tree performing similarly to the Random Tree for most instances.

A potential explanation for this is that the usage of random keys in Permutation GOMEA and the repair in qGOMEA, in combination with the random nature of Taillard's instances, may not contain any strong dependencies for which the univariate factorization is inadequate. As any tree-based FOS contains the univariate factorization, no performance difference between the Random Tree and Linkage Tree would be expected.

Furthermore, using a Random Tree and mutation aids in exploration in exchange for reduced exploitation, i.e. performing variation on correlated parameters.

We note that the instances with a larger number of jobs seem to be easier given the same number of machines and fewer evaluations (as listed in Table 2). This could be caused by a machine being a bottleneck. If a machine is a bottleneck all machines before it are less important: the processing times for these tasks do not appear in the critical path and are not reflected in the objective.

This does provide additional difficulties for black-box model-based approaches, such as qGOMEA. With the available information it could be very difficult to infer important structure using linkage learning. Providing additional information, for example through multi-objectivization – using additional secondary objectives, could provide a way forward here.

Comparing against state-of-the-art approaches from literature provides other insights. For instances with few machines, performance is comparable, whereas the state-of-the-art approaches HGM-EDA and DEP perform better on instances with larger numbers of machines. As having more machines may cause more orders to incur waiting time on the critical path, the potential interactions between orders may also span a larger distance. This causes a misalignment between the repair and encoding discussed previously, as well as the subsets found by using a Linkage Tree.

An important difference that should be noted however, is that we have not hybridized pGOMEA nor the newly proposed qGOMEA with a local searcher that is known to work well on PFS such as VNS4, as is the case for HGM-EDA and DEP. Moreover, the initialization for the state-of-the-art approaches employs problem-specific heuristics to seed the population with better solutions. As the hybrid HGM-EDA outperforms the GOMEA variants, while the non-hybrid GM-EDA does not, hybridizing GOMEA in such ways may well also result in considerable additional performance improvements. Such hybridization is however outside the scope of our article.

Another point of note is that comparing the Evolutionary Algorithms used themselves based on the results as reported here is difficult due to differences in employed settings. We have not tuned any algorithm to perform best on any problem or problem class in particular. The parameters for HGM-EDA and DES were however tuned on similar scheduling instances. The GOMEAs are designed with as few parameters as possible. A specific example of this is the population size. The population size in the variants of GOMEA as we have used them here, is controlled online using a population sizing scheme. While this scheme may eventually provide a better population size than a fixed value, especially without a (low) limit on the evaluations budget, it introduces an overhead compared to determining a good value beforehand. However, such pre-tuning of parameters (to specific problem instances or classes) may not always be possible in real-world practice.

Altogether, it is therefore hard to ascertain whether the observed improved performance for instances with a larger number of machines is due to the EA performing more effective variation or due to the use of specialized additional operators for the PFS problem.

6.3. Quadratic Assignment Problem

The results for the Quadratic Assignment Problem benchmark are listed in Table 4.

The original permutation GOMEA has noticeably worse performance for this problem. We notice that the use of random keys negatively im-

Table 3
Results for a subset of PFS instances (with index 1 & 2) as gap in percentage deviation for the Makespan criterion.

n	m	idx	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(a)	(b)	(c)	(d)
20	**5**	**1**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00	0.00	0.00	0.00
20	**5**	**2**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00	0.07	0.00	0.00
20	**10**	**1**	0.19	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00	0.51	0.00	0.00
20	**10**	**2**	0.06	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00	0.66	0.00	0.00
20	**20**	**1**	0.26	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.02	18.94	0.00	0.00
20	**20**	**2**	0.29	0.26	0.10	0.10	**0.00**	**0.00**	**0.00**	**0.00**	0.00	35.59	0.00	0.00
50	**5**	**1**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00	0.29	0.00	0.00
50	**5**	**2**	0.07	0.14	0.07	0.14	**0.00**	**0.00**	0.07	**0.00**	0.10	0.42	0.00	0.00
50	**10**	**1**	1.81	2.35	**1.12**	1.71	**1.12**	**1.12**	**1.12**	**1.12**	0.46	1.22	0.00	0.00
50	**10**	**2**	1.93	1.95	1.04	1.51	**0.62**	**0.68**	1.41	**0.50**	1.47	1.81	0.45	0.33
50	**20**	**1**	1.57	2.33	**1.18**	1.71	**1.10**	**1.18**	**1.17**	1.21	1.19	2.14	0.49	0.59
50	**20**	**2**	1.70	2.19	**0.54**	1.62	**0.56**	**0.71**	**0.71**	**0.67**	1.93	3.18	0.27	0.08
100	**5**	**1**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00	0.04	0.00	0.00
100	**5**	**2**	0.13	0.28	0.13	0.28	**0.00**	**0.00**	**0.00**	**0.00**	0.13	0.32	0.00	0.13
100	**10**	**1**	0.52	0.52	0.20	0.47	0.02	**0.00**	0.16	0.02	0.19	0.78	0.00	0.10
100	**10**	**2**	0.78	0.94	0.24	0.78	0.24	**0.00**	**0.00**	**0.00**	0.30	0.71	0.06	0.24
100	**20**	**1**	2.70	3.93	1.94	2.93	**1.61**	**1.51**	**1.52**	**1.55**	2.18	2.50	0.58	0.06
100	**20**	**2**	2.63	3.22	1.25	2.82	**1.10**	**1.13**	**1.12**	**1.05**	2.34	1.84	0.66	0.13
200	**10**	**1**	0.21	0.80	0.21	0.80	**0.09**	**0.09**	**0.09**	**0.09**	0.16	0.33	0.00	0.31
200	**10**	**2**	0.74	1.07	0.48	1.01	0.19	**0.16**	0.24	**0.15**	0.47	0.45	0.03	0.79
200	**20**	**1**	1.94	3.52	1.53	2.53	**1.23**	**1.17**	**1.18**	**1.17**	1.98	1.46	0.56	0.24
200	**20**	**2**	2.25	4.60	1.76	3.31	1.52	**1.30**	1.62	1.48	2.71	1.59	0.40	0.09
500	**20**	**1**	1.51	3.67	1.07	2.42	0.84	**0.67**	0.86	0.72	1.15	3.38	0.23	0.02
500	**20**	**2**	1.79	3.49	1.21	2.26	0.80	**0.72**	0.79	**0.59**	0.83	2.77	0.18	0.05

(1) RKSGA **(2)** IPSGA - CX **(3)** IPSGA - OX **(4)** IPSGA - PMX **(5)** Permutation GOMEA - LT **(6)** Permutation GOMEA - RT **(7)** qGOMEA - LT **(8)** qGOMEA - RT The following approaches are from literature and have been copied from [27] for reference. As no distributional data was available we could not perform a statistical test. **(a)** IG [28] **(b)** GM-EDA [29] **(c)** HGM-EDA [29] **(d)** DEP [27].

Table 4
Results for a subset of QAP instances, with a time limit of 10 minutes each, as gap in percentage deviation.

instance	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
chr22b	2.99	**0.00**	4.30	1.15	4.40	5.56	2.16	0.77	1.34	0.98
els19	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
esc128	53.95	3.03	27.27	**0.00**	26.35	20.98	**0.00**	**0.00**	**0.00**	**0.00**
esc32b	15.14	8.70	12.50	4.35	16.82	12.50	**0.00**	**0.00**	**0.00**	**0.00**
kra30b	0.32	0.25	0.25	0.13	0.73	0.70	0.23	**0.00**	**0.00**	**0.00**
lipa30b	**0.00**	**0.00**	11.85	**0.00**	14.24	13.83	10.83	**0.00**	**0.00**	**0.00**
lipa90a	1.52	0.91	1.09	0.84	1.11	1.04	0.98	0.70	0.75	**0.69**
lipa90b	21.77	18.82	19.75	18.41	19.84	19.54	19.12	**17.63**	17.86	**17.49**
nug30	0.79	**0.07**	1.21	0.44	1.35	1.43	1.43	**0.00**	0.08	**0.03**
rou20	1.35	0.20	1.08	0.20	1.51	1.26	1.24	**0.00**	**0.00**	**0.00**
scr15	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
scr20	0.03	**0.00**	0.03	**0.00**	1.96	0.96	0.05	**0.00**	**0.00**	**0.00**
sko64	1.07	**1.00**	2.95	1.64	3.98	3.58	3.19	**0.79**	0.93	**0.73**
ste36c	1.25	**0.00**	5.95	0.43	3.85	2.50	1.52	**0.16**	**0.28**	0.25
tai100b	3.24	**0.81**	5.98	1.51	8.30	7.37	5.01	**0.94**	1.50	**0.90**
tai256c	1.01	2.69	1.74	1.56	0.63	**0.45**	0.67	0.56	**0.52**	**0.51**
tai35a	9.94	3.39	5.55	3.44	5.40	4.76	4.57	**1.88**	2.18	**1.63**
tho150	4.23	4.30	7.06	4.24	6.90	6.06	4.73	1.82	2.18	**1.58**
tho40	1.45	**0.32**	3.54	1.45	3.92	2.86	3.31	**0.46**	0.93	0.58
wil50	0.43	**0.07**	2.01	0.52	1.58	1.15	1.08	0.26	0.26	0.20

(1) RKSGA **(2)** IPSGA - CX **(3)** IPSGA - OX **(4)** IPSGA - PMX **(5)** Permutation GOMEA - LT **(6)** Permutation GOMEA - RT **(7)/(8)** qGOMEA - LT - OX/PMX **(9)/(10)** qGOMEA - RT - OX/PMX

pacts performance here: after crossover the position is often off by one. A random key encodes a position dependent on the other keys. This yields behavior similar to the OX-crossover, which has similarly bad performance on this problem.

Furthermore, the current application of linkage learning is ineffective for this problem, with the RT notably outperforming the LT variants of both Permutation GOMEA and qGOMEA.

The cause of these two issues is likely the same. The QAP is not a problem in which the order of positions is necessarily meaningful: the order of the positions in an instance can be changed by permuting the distance matrix, and can therefore be completely arbitrary. Using distance as a dependency metric can therefore yield a completely arbitrary FOS, which stays relatively fixed.

Conversely, qGOMEA performs remarkably well when using a random tree model, being the top performer in most of the instances. Unlike the previously discussed linkage tree, the resulting FOS changes constantly. As the QAP assigns values to all pairs, randomly considering pairs to be correlated may be a good fit. It gives equal attention to all pairs, resulting in more variation.

Due to the good performance of the IPSGA using the PMX crossover, we additionally report the results of using a similar style of repair for qGOMEA. In this case the crossover with the virtual donor is performed

Table 5
Results for OAS, as gap in percentage deviation, where $n = 100$, index = 1.

t	R	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
1	1	1.11	2.41	1.30	1.67	0.97	1.02	**0.46**	0.74
1	3	1.22	2.03	1.22	1.62	1.22	1.32	**0.71**	**0.96**
1	5	**0.17**	0.68	0.26	0.43	0.43	0.56	**0.17**	0.43
1	7	**0.00**	0.01	**0.00**	**0.00**	**0.00**	0.00	**0.00**	**0.00**
1	9	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
3	1	1.03	1.96	1.18	1.57	1.08	1.27	**0.49**	1.03
3	3	1.18	2.51	1.27	2.03	1.13	1.27	**0.61**	**0.80**
3	5	1.09	1.96	1.38	1.62	1.18	1.38	**0.80**	1.38
3	7	**0.17**	0.76	0.17	0.68	0.17	0.51	**0.17**	**0.17**
3	9	0.11	0.64	0.18	0.64	0.18	0.37	**0.00**	0.23
5	1	1.76	3.15	1.76	3.03	1.48	1.76	**0.78**	1.03
5	3	1.47	2.98	1.47	3.33	1.37	1.57	**0.77**	**0.77**
5	5	1.24	3.48	0.96	2.91	0.86	1.05	**0.38**	0.67
5	7	0.68	1.84	0.73	2.08	0.48	0.73	**0.19**	0.77
5	9	1.81	4.97	1.81	4.69	1.40	1.72	**0.90**	1.22
7	1	1.99	5.10	1.37	6.58	1.37	1.80	**0.80**	1.28
7	3	1.61	6.60	1.26	5.14	1.35	1.26	**0.55**	1.08
7	5	1.99	14.48	1.69	10.59	1.74	1.39	**0.78**	**0.78**
7	7	3.65	16.96	3.77	13.42	3.11	2.50	**1.58**	1.92
7	9	2.68	11.95	3.41	8.68	3.07	2.53	**1.61**	**1.80**
9	1	2.24	41.78	1.27	29.05	1.52	1.42	**0.98**	1.08
9	3	3.59	38.14	1.48	27.45	2.15	1.69	**0.86**	**0.97**
9	5	3.68	32.76	2.84	22.46	2.19	1.40	0.69	**0.52**
9	7	1.92	33.61	1.43	23.27	1.63	1.10	**0.58**	**0.57**
9	9	3.00	38.69	2.91	28.25	2.51	1.00	**0.24**	**0.24**

(1) RKSGA **(2)** IPSGA - CX **(3)** IPSGA - OX **(4)** IPSGA - PMX **(5)** Permutation GOMEA - LT **(6)** Permutation GOMEA - RT **(7)** qGOMEA - LT **(8)** qGOMEA - RT.

like in PMX, as opposed to OX. The performance of this configuration shows the best performance across all evaluated approaches. Most notably, the gap between random tree and linkage tree shrinks considerably, in many cases making any difference statistically insignificant.

6.4. Order Acceptance and Scheduling

The results for the Order Acceptance and Scheduling benchmark are listed in Table 5.

For Order Acceptance and Scheduling it is known that instances with higher τ and lower R are more difficult to optimize [12]. A similar trait can be found here. For example the instance with $\tau = 1$ and $R = 9$ is solved to optimality with all approaches, whereas a large difference in performance can be seen for $\tau = 9$ and $R = 1$.

Having the right dependency metric, crossover and encoding is important to gain an increase in performance when using a Linkage Tree. Permutation GOMEA does not benefit from the usage of the Linkage Tree, with the Random Tree showing equivalent results. This is not the case for qGOMEA, where utilizing linkage provides better results for a large portion of the instances. qGOMEA outperforms on this class of problem structure where relative offsets of elements in close proximity are important.

7. Discussion

In this section we discuss and reflect on the choices made and the limitations of this work. In particular, computational resources are limited. This has led to some experiments being limited in number of time and repetitions, as well as limitations in the configurations evaluated. Furthermore, there are many potential points of improvement that were left uninvestigated, which we highlight as future work.

Among the configurations evaluated for the SimpleGAs, the EA utilizing the ER crossover was dropped from the result tables due to bad results. This is likely caused by the ER crossover's assumptions: being neighbors is important, but not the order itself. It is commonly applied to symmetric TSP, yet none of the evaluated problems in this work are symmetric. A comparison with a symmetric TSP problem would likely be a better comparison for this operator.

One example of limitations regarding the number of configurations is that we only covered a configuration of qGOMEA using PMX-style repair once. We note that such modifications to qGOMEA can improve performance on various problems, and much like traditional crossover, provide significant performance benefits when the chosen repair matches the problem at hand. In this case PMX with SimpleGAs performed better on the QAP, leading to the potential of the repair performing better as well. We did not evaluate the performance of the aforementioned modification on other problems, and as such we cannot confirm whether this pattern holds in general, even though we expect it to.

Tangentially related is the fact that the introduction of this choice leads to qGOMEA having a parameter which is dependent on the problem at hand. This is a step backwards from the usual automatic usability that GOMEA provides for binary problems. We did not investigate automating the choice of repair operator.

As qGOMEA is a departure from Permutation GOMEA in various ways, it is useful to determine the effectiveness of its components, beyond just linkage learning as we have done in this work. For example, which operators are key to solving particular problems and instances? Such insight would be useful to obtain future improvements.

Using distance as linkage metric is not universally applicable to permutation problems. It assumes that values assigned to nearby positions are related, i.e. positions can be assumed to be ordinal and provided in order. This is not true in general, as is illustrated by the Quadratic Assignment Problem and the corresponding results. It is possible to use an alternative metric, such as Mutual Information as used in [5]. Utilizing Mutual Information in permutation space directly comes with the trade-off. This metric may require a significantly larger population to obtain sufficient information to build a good model. Of special note is that it infers linkage due to the uniqueness constraint, which most permutation-based operators already account for.

If a particular kind of disruption is avoided entirely by the recombination operator already accounting for it, Linkage Learning no longer needs to account for it either. As such linkage should be adapted to the specific kind of structure an operator is recombining to reach the highest efficiency possible.

Additionally, using a Family of Subsets to model linkage and dependencies may be difficult in a general setting. An example of this is the strong linkage introduced by the uniqueness constraint, which encompasses all variables. Yet many problems contain sparsity utilizable by considering subsets of variables. Consider an instance of Traveling Salesperson with cities, each having places to visit. An efficient route will group places within these cities together, which can be represented by a subset in the Family of Subsets. Combined with the right operators, such as the ones in qGOMEA, this should provide the ability to both optimize intercity tours and inner-city tours. As another example, a scheduling problem such as Order Acceptance and Scheduling has release times and deadlines. Such a problem naturally has particular objects (i.e. orders) only appear in particular positions. Reducing the impact of the uniqueness constraint and again giving rise to certain more closely related subsets.

In many practical problems there is a notion of such structure. If the Linkage Learning can recognize this structure and has the right operators in place to preserve such structure under recombination, then the GOMEA family approaches is expected to do well. If there is a misalignment i.e. a problem has no such structure that can be recognized, performance is expected to be worse, and other approaches will likely do better.

As qGOMEA's operators utilize the sequence-like property that many permutation problems have, the applicability of qGOMEA on a wider range of permutation problems is limited. An alternative would be to work with a more general method of recombination, such as the algebraic operations and lattice based operations described in [30,31]. For such operators linkage between variables could, for example, be utilized by learning a pruned generating set and lattice from the population, or even learn linkage between algebraic operations.

A closely related topic is the encoding used to represent a solution. In this article we have considered both the use of random keys and the straightforward encoding using integers. In both cases one can also optimize over the inverse permutation instead. Depending on the problem and approach used, doing so may align better with the problem's structure and increase performance. While we have ensured usage of the best performing representation for all approaches and problems in general, as swapping the matrices in a QAP instance inverts the representation used implicitly, no such representation exists for the Quadratic Assignment Problem. As QAP was included in order to assess performance in a setting where assumptions are potentially violated, rather than solving the instances themselves, we did not extensively investigate per-instance representations. However, in various, though limited, additional runs that we performed on both Order Acceptance and Scheduling and Permutation Flowshop, we did observe that using the inverse permutation encoding substantially reduced performance. Still, it may be of interest to investigate the use of both representations simultaneously in a future work. For example by determining linkage through the combination of both representations, or by using a recombination operator on the alternative representation.

In this work we have been restricted in a black-box setting with full evaluations for all approaches. As random keys are not used in qGOMEA, the decoding step is not required; allowing for re-evaluation in a fashion similar to discrete Cartesian problems. The GOMEA family is most commonly evaluated in a grey-box setting where partial re-evaluations are performed, with a corresponding gain in scalability and performance with respect to time [32]. Yet this change makes it hard to compare approaches such as the SimpleGAs, and requires significant changes in order to facilitate caching for certain problems. We therefore recommend a grey-box approach for future work which aims to provide significant improvements for a specific problem.

Similarly, we have not cached any evaluations (i.e. Long Term Memory Assistance). During convergence, it is likely that the same solution is evaluated more than once. Caching the result of an evaluation could save time regarding evaluating these solutions. Furthermore, if only saving time this way is not enough, employing a surrogate model as in [33] could provide a solution.

The computational resources required in the case of Permutation Flowshop are especially notable, with evaluation budgets larger than 100,000,000 leading to each run taking hours. It is of interest to note that Taillard benchmark problems are known to have little structure. It may be of interest to run experiments on Permutation Flowshop instances which have more structure to exploit such as in [10]. Such instances may be more representative of real-world situations. Moreover, by utilizing this structure, such instances can likely be solved more efficiently, with a lower budget of evaluations.

Finally, given the conditional nature of permutations and the overlapping nature of the constraints and most problem formulations; another promising approach is the use of the recently introduced Conditional Linkage Models [34]. It is possible that similar scalability benefits are obtained for permutation problems as well.

8. Conclusion

In this work we revisited the design of a GOMEA for permutation spaces. We found that solving permutation problems using GOMEA, specialized operators can be significantly more effective than using standard operators and a random-key encoding. This is caused by the mismatch between the kinds of structure that random keys preserve – namely relative ordering – and the structure of many other permutation problems in practice – being sequential and absolute position.

We note the most significant improvements on the QAP problem, in which even the SimpleGAs that we considered were capable of providing better quality solutions. Yet, this improvement does not come at a cost of performance for the other problems that we considered, showing equal or improved performance. Furthermore, qGOMEA's operators have shown to be capable of exploiting structure in various real-world problems as well as benchmark problems, contrary to the other evaluated approaches.

As is mostly known, but clearly found here, good performance of an approach on a benchmark function does not necessarily transfer to a real-world problem. More positively, qGOMEA's mixing operators seem to generalize well to a large suite of permutation problems, even without the assistance of linkage learning.

The theory behind linkage learning states that the number of disruptions of important building blocks need to be minimized. Much like one would expect on basis of this theory, we find that not every operator is a good fit for every problem. While OX and random keys show good performance on PFS and OAS, QAP is notably different, with PMX and CX outperforming OX and random keys. Ideally linkage learning in GOMEA can generalize this choice of operator. Yet; we find that the current form of Linkage Learning does not provide the expected increase in performance in general, with only showing a large improvement in the Sequential Pairs variant of the Inversion benchmark, as well as a smaller improvement for the OAS problem.

Furthermore, the currently used sources for linkage information are not necessarily good descriptors of linkage for a general permutation problem. It is clear that they make too many assumptions: having two positions being next to one another has little meaning in the context of the Quadratic Assignment Problem. Defining such a universal descriptor is still an open question. Answering this question may be hindered by a potential lack of information in a black-box context. Nonetheless, when a good descriptor is present, improved performance is observed.

We conclude that compared to common crossover operators on random keys in pGOMEA, the introduction of the specialized operators in qGOMEA are beneficial on many problems. This is most notable with respect to QAP instances, where the original permutation GOMEA was among the worst contenders. As such we believe that qGOMEA is a useful addition to the GOMEA family.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Arthur Guijt: Conceptualization, Methodology, Software, Investigation, Writing – original draft, Writing – review & editing. **Ngoc Hoang Luong:** Investigation, Writing – original draft. **Peter A.N. Bosman:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision. **Mathijs de Weerdt:** Conceptualization, Methodology, Writing – review & editing, Supervision.

Acknowledgment

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

[1] Z. Michalewicz, D.B. Fogel, An Evolutionary Approach – How to Solve It: Modern Heuristics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, doi:10.1007/978-3-662-07807-5.
[2] D. Thierens, Scalability problems of simple genetic algorithms, Evol. Comput. 7 (4) (1999) 331–352, doi:10.1162/evco.1999.7.4.331.
[3] J.A. Lozano, P. Larrañaga, I. Inza, E. Bengoetxea, Towards a New Evolutionary Computation. Advances in Estimation of Distribution Algorithms, Springer-Verlag, Berlin, 2006, doi:10.1007/3-540-32494-1.
[4] M. Pelikan, K. Sastry, E. Cantú-Paz, Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications, Springer-Verlag, Berlin, 2006, doi:10.1007/978-3-540-34954-9.

[5] D. Thierens, P.A.N. Bosman, Optimal mixing evolutionary algorithms, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, 2011, pp. 617–624, doi:10.1145/2001576.2001661.

[6] A. Bouter, T. Alderliesten, C. Witteveen, P.A.N. Bosman, Exploiting linkage information in real-valued optimization with the real-valued gene-pool optimal mixing evolutionary algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 705–712, doi:10.1145/3071178.3071272.

[7] N.H. Luong, H. La Poutré, P.A.N. Bosman, Exploiting linkage information and problem-specific knowledge in evolutionary distribution network expansion planning, Evol. Comput. 26 (3) (2018) 471–505, doi:10.1162/evco_a_00209. PMID: 28388221

[8] P.A.N. Bosman, N.H. Luong, D. Thierens, Expanding from discrete cartesian to permutation gene-pool optimal mixing evolutionary algorithms, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, 2016, pp. 637–644, doi:10.1145/2908812.2908917.

[9] J.C. Bean, Genetic algorithms and random keys for sequencing and optimization, ORSA J. Comput. 6 (2) (1994) 154–160, doi:10.1287/ijoc.6.2.154.

[10] G.H. Aalvanger, N.H. Luong, P.A.N. Bosman, D. Thierens, Heuristics in permutation GOMEA for solving the Permutation Flowshop scheduling problem, in: Proceedings of the International Conference on Parallel Problem Solving from Nature, Springer, 2018, pp. 146–157, doi:10.1007/978-3-319-99253-2_12.

[11] R.E. Burkard, S.E. Karisch, F. Rendl, Qaplib–a Quadratic Assignment Problem library, J. Glob. Optim. 10 (4) (1997) 391–403, doi:10.1023/A:1008293323270.

[12] B. Cesaret, C. Oğuz, F.S. Salman, A tabu search algorithm for order acceptance and scheduling, Comput. Oper. Res. 39 (6) (2012) 1197–1205, doi:10.1016/j.cor.2010.09.018.

[13] C. Oğuz, F.S. Salman, Z.B. Yalçın, et al., Order acceptance and scheduling decisions in make-to-order systems, Int. J. Prod. Econ. 125 (1) (2010) 200–211, doi:10.1016/j.ijpe.2010.02.002.

[14] L. He, A. Guijt, M. de Weerdt, L. Xing, N. Yorke-Smith, Order acceptance and scheduling with sequence-dependent setup times: a new memetic algorithm and benchmark of the state of the art, Comput. Ind. Eng. 138 (2019) 106102, doi:10.1016/j.cie.2019.106102. http://www.sciencedirect.com/science/article/pii/S0360835219305716

[15] L. Davis, Applying adaptive algorithms to epistatic domains, in: Proceedings of the IJCAI, volume 85, 1985, pp. 162–164.

[16] D.E. Goldberg, R. Lingle, et al., Alleles, loci, and the traveling salesman problem, in: Proceedings of an International Conference on Genetic Algorithms and Their Applications, volume 154, Lawrence Erlbaum, Hillsdale, NJ, 1985, pp. 154–159.

[17] I.M. Oliver, D. Smith, J.R.C. Holland, Study of permutation crossover operators on the traveling salesman problem, in: Proceedings of the Second International Conference on Genetic Algorithms: Genetic Algorithms and Their Applications: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA, Hillsdale, NJ: L. Erlhaum Associates, 1987., 1987.

[18] L.D. Whitley, T. Starkweather, D. Fuquay, Scheduling problems and traveling salesmen: the genetic edge recombination operator, in: Proceedings of the ICGA, volume 89, 1989, pp. 133–140.

[19] L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.

[20] H. Kargupta, K. Deb, D.E. Goldberg, Ordering genetic algorithms and deception, in: Proceedings of the PPSN, Citeseer, 1992, pp. 49–58.

[21] E. Taillard, Benchmarks for basic scheduling problems, Eur. J. Oper. Res. 64 (2) (1993) 278–285, doi:10.1016/0377-2217(93)90182-M.

[22] S.N. Chaurasia, A. Singh, Hybrid evolutionary approaches for the single machine order acceptance and scheduling problem, Appl. Soft Comput. 52 (2017) 725–747, doi:10.1016/j.asoc.2016.09.051.

[23] J. Ceberio, E. Irurozki, A. Mendiburu, J.A. Lozano, A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem, IEEE Trans. Evol. Comput. 18 (2) (2013) 286–300, doi:10.1109/TEVC.2013.2260548.

[24] E. Taillard, Eric taillards page, 2015, (http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html), [Online; accessed 17-March-2020].

[25] R.E. Burkard, E. Cela, S.E. Karisch, F. Rendl, Qaplib - a Quadratic Assignment Problem library, (http://anjos.mgi.polymtl.ca/qaplib/inst.html), [Online; accessed 17-March-2020].

[26] Y.L.T.V. Silva, A. Subramanian, A.A. Pessoa, Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times, Comput. Oper. Res. 90 (2018) 142–160.

[27] V. Santucci, M. Baioletti, A. Milani, Solving Permutation Flowshop scheduling problems with a discrete differential evolution algorithm, AI Commun. 29 (2) (2016) 269–286, doi:10.3233/AIC-150695.

[28] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the Permutation Flowshop scheduling problem, Eur. J. Oper. Res. 177 (3) (2007) 2033–2049, doi:10.1016/j.ejor.2005.12.009.

[29] J. Ceberio, E. Irurozki, A. Mendiburu, J.A. Lozano, A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem, IEEE Trans. Evol. Comput. 18 (2) (2014) 286–300, doi:10.1109/TEVC.2013.2260548.

[30] M. Baioletti, G. Di Bari, A. Milani, V. Santucci, An experimental comparison of algebraic crossover operators for permutation problems, Fundam. Inform. 174 (3–4) (2020) 201–228, doi:10.3233/FI-2020-1940.

[31] V. Santucci, M. Baioletti, A. Milani, An algebraic framework for swarm and evolutionary algorithms in combinatorial optimization, Swarm Evol. Comput. 55 (2020) 100673, doi:10.1016/j.swevo.2020.100673.

[32] A. Bouter, T. Alderliesten, A. Bel, C. Witteveen, P.A.N. Bosman, Large-scale parallelization of partial evaluations in evolutionary algorithms for real-world problems, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2018, pp. 1199–1206, doi:10.1145/3205455.3205610.

[33] A. Dushatskiy, A.M. Mendrik, T. Alderliesten, P.A.N. Bosman, Convolutional neural network surrogate-assisted GOMEA, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 753–761, doi:10.1145/3321707.3321760.

[34] A. Bouter, S.C. Maree, T. Alderliesten, P.A.N. Bosman, Leveraging conditional linkage models in gray-box optimization with the real-valued gene-pool optimal mixing evolutionary algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2020, pp. 603–611, doi:10.1145/3377930.3390225.