Expanding from Discrete Cartesian to Permutation Gene-pool Optimal Mixing Evolutionary Algorithms

Peter A.N. Bosman
Centrum Wiskunde & Informatica (CWI)
P.O. Box 94079
1090 GB Amsterdam
The Netherlands
Peter.Bosman@cwi.nl

Ngoc Hoang Luong
Centrum Wiskunde & Informatica (CWI)
P.O. Box 94079
1090 GB Amsterdam
The Netherlands
Hoang.Luong@cwi.nl

Dirk Thierens
Dept. of Computer Science
Utrecht University
P.O. Box 80089 3508 TB
Utrecht
The Netherlands
D.Thierens@uu.nl

ABSTRACT

The recently introduced Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) family, which includes the Linkage Tree Genetic Algorithm (LTGA), has been shown to scale excellently on a variety of discrete, Cartesian-space, optimization problems. This paper shows that GOMEA can quite straightforwardly also be used to solve permutation optimization problems by employing the random keys encoding of permutations. As a test problem, we consider permutation flowshop scheduling, minimizing the total flow time on 120 different problem instances (Taillard benchmark). The performance of GOMEA is compared with the recently published generalized Mallows estimation of distribution algorithm (GM-EDA). Statistical tests show that results of GOMEA variants are almost always significantly better than results of GM-EDA. Moreover, even without using local search, the new GOMEA variants obtained the best-known solution for 30 instances in every run and even new upper bounds for several instances. Finally, the time complexity per solution for building a dependency model to drive variation is an order of complexity less for GOMEA than for GM-EDA, altogether suggesting that GOMEA also holds much promise for permutation optimization.

CCS Concepts

•Mathematics of computing → Evolutionary algorithms; •Computing methodologies → Planning and scheduling; Search methodologies;

Keywords

Genetic Algorithms; Estimation-of-Distribution Algorithms; Linkage learning; Optimal Mixing; Permutation Problems; Random Keys; Scheduling

GECCO '16, July 20-24, 2016, Denver, CO, USA

DOI: http://dx.doi.org/10.1145/2908812.2908917

1. INTRODUCTION

The need to exploit problem structure in evolutionary algorithms (EAs) to ensure efficient performance has long been a cornerstone of the field, from its theoretical underpinnings that show that in the worst case, exponential-time scale-up is obtained for problems that are polynomial-time solvable (see, e.g., [16]), to the building of specific variation operators for specific problems (see, e.g., [11]), to the design of methods and techniques capable of detecting and exploiting problem structure automatically, *during* optimization (see, e.g., [4, 14]). The latter is a key characteristic of a specific type of EAs, Estimation-of-Distribution Algorithms (EDAs), that estimate probability distributions and subsequently sample these distributions to generate new solutions, thereby respecting the structure captured in the estimated distribution (see, e.g., [14]). EDAs can be seen as a specific interpretation of the more general notion of model-based EAs (MBEAs) in which a model is used to drive variation. Such models, that need not be probabilistic, can be tuned during optimization. Especially this property makes MBEAs particularly important when considering optimization problems that require taking a Black Box Optimization (BBO) perspective. In BBO, little is assumed to be known about the problem being solved, except, e.g., the domain of the variables. Many real-world problems, when modeled to a practically-relevant level of detail, are of this form, sometimes providing only simulations for objective functions. Making the plausible assumption then that the problem being solved has some exploitable structure, finding and exploiting this structure leads to more efficient optimization than when operators are used that do not exploit problem structure.

Most research on MBEAs is focused on either discrete variables or continuous (real-valued) variables. Moreover, the search spaces are considered to be Cartesian, i.e. each variable has its own domain and can be assigned any value from that domain. Interestingly, state-of-the-art MBEAs (e.g., hBOA [13], DMSGA-II [10] and LTGA/LT-GOMEA [4] for the discrete case and AMaLGaM [2], NES [17], and CMA-ES [7] for the continuous case) are often considered to be the most powerful EAs for their respective domains, being able to efficiently solve a large class of optimization problems that exhibit some form of exploitable structure. Many real-world optimization problems are however *permutation-based*, particularly in logistics such as (vehicle) routing, timetabling,

and scheduling. These are important, societally-relevant problems that may well require taking a BBO perspective when modeled to include (many) real-world details. The solution space for these problems however is non-Cartesian, being formed by all permutations of length ℓ, where ℓ is the number of problem variables. Consequently, specialized variation operators or solution encodings are required that ensure that offspring solutions are permutations.

Likely due to the added difficulty of permutation spaces being non-Cartesian, there are considerably fewer results on MBEAs for permutation problems. The Generalized Mallows Estimation-of-Distribution Algorithm (GM-EDA) is a a recent, state-of-the-art MBEA [5]. The GM-EDA uses a model that could be considered the Gaussian-distribution equivalent in permutation space. To estimate it, a central permutation and a notion of variance around this permutation needs to be computed. Sampling then ensures that new solutions are permutations. Having a probabilistic model that is specifically meant for permutations allows the GM-EDA to outperform earlier permutation MBEAs that are mostly (ad-hoc) adjusted EDAs originally designed for discrete or continuous real-valued Cartesian search spaces. A downside of GM-EDA is that, similar to many state-of-the-art EDAs, the computational complexity of building and sampling the model every generation is relatively high. Specifically, it is $\mathcal{O}(\ell^2 n)$ where n is the population size. Because n solutions are generated and evaluated every generation, the computational complexity per solution is $\mathcal{O}(\ell^2)$.

In this paper, we focus on a specific type of MBEA that was recently introduced: the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) family, which includes the Linkage Tree Genetic Algorithm (LTGA) [4]. Different from EDAs, models in this type of MBEA are focused at capturing *linkage* information, which is often also called dependency information, i.e. a notion of which variables together have a positive synergistic effect on fitness and therefore should be treated jointly during variation to prevent disrupting important partial solutions that have been evolved so far. Especially when the linkage tree is used for the linkage model in GOMEA, state-of-the-art performance is obtained on a variety of discrete, Cartesian-space, optimization problems, including well-known benchmark problems and real-world applications [4, 12]. Another advantage is that powerful linkage models can be built at relatively low computational cost. For instance, the computational complexity of building a linkage tree is $\mathcal{O}(\ell^2 n)$. Although this is the same complexity as the GM-EDA requires, in GOMEA linkage information is exploited far more intensively, leading to $\mathcal{O}(\ell n)$ function evaluations per generation. Thus, the model-related computational complexity per evaluated solution is actually an order of complexity less: $\mathcal{O}(\ell)$.

The advantages of GOMEA are interesting, but so far EAs in this family have only been designed and used for discrete Cartesian search spaces. Therefore, an interesting question is whether the GOMEA family can be expanded to also efficiently tackle problems in permutation spaces. Here, we take a first approach to achieving this using a well-known encoding of permutations, called *random keys*, that was introduced to ensure that classical, well-known crossover operators always generate feasible permutations [1]. Since random keys form a Cartesian search space, we find that GOMEA can be combined quite straightforwardly with this encoding to solve permutation optimization problems.

The remainder of this paper is organized as follows. In Section 2 we briefly summarize GOMEA for discrete Cartesian spaces, after which, in Section 3, we indicate what changes we applied to make GOMEA work well with random keys. In Section 4 we then briefly discuss a well-known permutation optimization problem that we use in our experiments: permutation flowshop scheduling. We compare the results of our newly designed GOMEA variants with GM-EDA in Section 5 and draw our final conclusions in Section 6.

2. GOMEA FOR CARTESIAN SPACES

As in any EA, a population $\mathcal{P}$ of $n = |\mathcal{P}|$ solutions is used in GOMEA. Typically, these solutions are initialized randomly, especially when taking a BBO perspective, but this is not required. In the following, we describe specific details of GOMEA, including a practical implementation that no longer requires manually setting the population size.

2.1 Encoding

Solutions $\boldsymbol{x} = (x_0, x_1, \ldots, x_{\ell-1})$ are encoded using ℓ discrete variables. The domain of variable i is $\mathbb{D}_i$ and the full search space is their Cartesian product, i.e. $\boldsymbol{x} \in \times_{i=0}^{\ell-1} \mathbb{D}_i$. A classical case is given by binary strings, i.e. $\mathbb{D}_i = \{0, 1\}$.

2.2 Model Building

In every generation, a model is built, based on $\mathcal{P}$, to drive variation. The model in GOMEA is a *linkage model* that captures dependencies between problem variables. Such dependencies are encoded using the mathematical concept of Family of Subsets (FOS), sometimes also referred to as the FOS model. Let L be the set of all variable indices, i.e., $L = \{0, 1, \ldots, l-1\}$. A FOS $\mathcal{F}$ is a set containing subsets of L, i.e. $\mathcal{F}$ is a subset of the powerset of L, i.e., $\mathcal{F} \subseteq \wp(L)$ and can be written as $\mathcal{F} = \{\boldsymbol{F}^0, \boldsymbol{F}^1, \ldots, \boldsymbol{F}^{|\mathcal{F}|-1}\}$, where $\boldsymbol{F}^i \subseteq \{0, 1, \ldots, l-1\}$. A subset $\boldsymbol{F}^i$, also called a *linkage set*, identifies variables that exhibit a degree of joint dependency and should thus be handled *together* during variation.

The most commonly used variant of the FOS model is the linkage tree (LT) (see Figure 1). The corresponding GOMEA variant, LT-GOMEA, is also known as the Linkage Tree Genetic Algorithm (LTGA) [4]. A key property that makes the LT-FOS a highly effective linkage model is that it captures both low-order dependencies and higher-order dependencies by hierarchically decomposing a notion of dependency between variables. Specifically, an LT can be learned by use of agglomerative, or bottom-up, hierarchical clustering. There exists an efficient implementation that combines 1) the so-called Unweighted Pair Group Method with Arithmetic Mean (UPGMA) that extends a notion of pairwise dependency to groups of variables, with 2) the so-called reciprocal nearest-neighbor chain technique, that builds an LT for ℓ variables from a data set of size n in $\mathcal{O}(\ell^2 n)$ time [6]. For details we refer the interested reader to the literature. Conceptually, $\mathcal{F}$ is initialized with ℓ singleton linkage sets $\boldsymbol{F}^i, i \in \{0, 1, \ldots, \ell-1\}$. Subsequently, a notion of dependency between linkage sets is calculated. The two most dependent linkage sets $\boldsymbol{F}^i$ and $\boldsymbol{F}^j$ are merged into a new linkage set $\boldsymbol{F}^i \cup \boldsymbol{F}^j$, which is then added to $\mathcal{F}$. The dependency values between the new linkage set and the remaining linkage sets are then updated. $\boldsymbol{F}^i$ and $\boldsymbol{F}^j$ are not discarded, but they are also not considered for merging anymore. The two most dependent linkage sets continue to be merged until a single set containing all variables has

been constructed. An LT contains $2\ell - 1$ linkage sets. In GOMEA, the notion of dependency between two variables that is used, is the information-theoretic concept of mutual information (for more details, see, e.g. [4]).

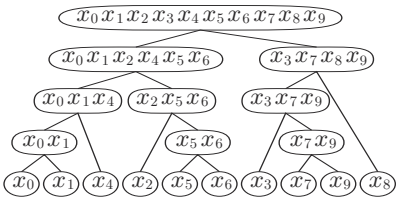

Figure 1: Example of a linkage tree for 10 variables.

2.3 Selection and Variation

Selection and variation are intertwined in a single operator called Gene-pool Optimal Mixing (GOM). GOM is used to transform *every* parent solution $\boldsymbol{p} \in \mathcal{P}$ into an offspring solution $\boldsymbol{o} \in \mathcal{O}$ through iterative mixing events that are guided by $\mathcal{F}$. Once every solution has been transformed, the current population $\mathcal{P}$ is replaced by the offspring $\mathcal{O}$.

For every parent solution $\boldsymbol{p}$, GOM has 2 phases. In phase 1, the FOS is iterated over in a random order (different order for every parent). For every linkage subset $\boldsymbol{F}^i$, a donor solution $\boldsymbol{d}$ is randomly picked from $\mathcal{P}$ and the values for the variables identified by $\boldsymbol{F}^i$ are copied from $\boldsymbol{d}$ to $\boldsymbol{p}$. If this results in solution $\boldsymbol{p}$ not becoming worse, the change is accepted, otherwise the change is reverted. Phase 2 is the so-called *forced-improvements* (FI) phase. Phase 2 is only entered if a solution was not changed during phase 1, or if the best solution in the population has not improved for a number of subsequent generations (called the no-improvement-stretch (NIS)). The latter is required because changes are also accepted in phase 1 if the solution maintains the same fitness level. To prevent GOMEA from becoming stationary on a fitness plateau, if the NIS is larger than $1+\lfloor^{10}\log(n)\rfloor$, phase 2 is also entered. Phase 2 is similar to phase 1, but instead of using a random donor, the best solution in the population is used. Moreover, now only strict fitness improvements are accepted and as soon as an improvement occurs, phase 2 is stopped. If at the end of phase 2 a solution still has not changed, the offspring becomes a copy of the best solution.

If, either in phase 1 or in phase 2, a linkage set is encountered that contains all variables (i.e., set L), such as the root node of the linkage tree, this set is disregarded because it implies that all problem variables should be copied together, which would yield no new offspring but would only correspond to additional diversity-reducing selection pressure.

2.4 Population Sizing Scheme

A crucial parameter of any EA is the population size n. If set too small, premature convergence occurs. If set too large, convergence takes overly long. Determining the optimal value of n to reach, e.g., the optimal value or a certain approximation ratio, is theoretically interesting, but does not solve the practical issue of how to best set n when tackling a new problem. Therefore, we employ a previously published scheme that prevents having to specify n by interleaving multiple instances of an EA with different population sizes [8]. Although perhaps somewhat counter-intuitive, when combined with GOMEA, the number of evaluations required to solve various benchmark problems is typically increased only by a factor of 1 to 4.

In this scheme, GOMEA is started with a small population size, n_{base}. Every 4^{th} generation, a single generation is executed of a GOMEA instance with a population size twice as large. Recursively, for every 4^{th} generation of that instance, a single generation is executed of yet another GOMEA instance with a population size again twice as large, and so on. This exponentially grows the largest population size over time, while still allowing GOMEA instances with smaller population sizes to converge faster, performing twice as many evaluations. In this paper, we use $n_{\text{base}} = 1$.

If at the end of a generation of any GOMEA instance, the average fitness is found to be smaller than the average fitness of a GOMEA instance with a larger population size, it is deemed to have been overtaken and it is stopped. Also, every other GOMEA instance with a smaller population size is then stopped. GOMEA instances are furthermore also stopped if no diversity remains (all solutions are the same).

Finally, whereas in the original implementation of this scheme all EA instances were completely independent, in GOMEA we make one small adjustment: the best-ever evaluated solution is shared by all populations and is used in phase 2 of GOM as the donor, rather than the best solution in the population. However, the conditional statement used to detect whether phase 2 should be entered, still uses the best solution in each individual population.

3. GOMEA FOR PERMUTATION SPACES USING RANDOM KEYS

A key strength of GOMEA is testing partial solutions within other solutions to see if improvements occur, making each mixing event optimal in its own right (hence the name optimal mixing). To maintain this strength when expanding to permutation spaces, either variation needs to be revised to sensibly work with permutations, or encoding needs to be revised so that variation does not need to change (much). In this paper, we consider the latter approach. In the following, we explain the required changes, as compared to Section 2.

3.1 Encoding

The random keys encoding for permutations is an encoding of permutations of $(0, 1, \ldots \ell - 1)$ in an ℓ–dimensional real-valued Cartesian space [1]. Specifically, the permutation encoded by ℓ real-valued variables $\boldsymbol{r} = r_0, r_1, \ldots, r_{\ell-1}$ is obtained by sorting them (in ascending order). In other words, the integer permutation $\boldsymbol{x}$ that is encoded by $\boldsymbol{r}$ is the permutation that corresponds to the sorting order of $\boldsymbol{r}$, i.e. such that $r_{x_0} < r_{x_1} < \ldots < r_{x_{\ell-1}}$ holds. For example, for $\boldsymbol{r} = (0.57, 0.93, 0.12, 0.43)$ we have $\boldsymbol{x} = (2, 3, 0, 1)$. A big advantage of random keys is that crossing over partial solutions always results in an encoding of a permutation, so any form of crossover, or in our case, optimal mixing, can straightforwardly be used. Although random keys may in principle take any value, we restrict their values to $[0, 1]$.

3.2 Model Building

To build a linkage tree with the method specified in Section 2.2 a numeric quantification of dependency between two variables is needed. The stronger the dependency, the larger the value should be. For permutations, we use a symmetric notion of dependency that is composed of two factors, i.e.:

$$\delta(j, i) = \delta(i, j) = \delta_1(i, j)\delta_2(i, j) \tag{1}$$

The first factor focuses on dependency in terms of so-called *relative-ordering information*. Intuitively, we need a notion of certainty about the ordering of two integers. For this, we use an information-theoretic measure, meant to convey similar information as the mutual information measure used in the discrete Cartesian case. Specifically, we compute the entropy of the probability that integer i appears before integer j, or equivalently, that random key r_i is smaller than random key r_j. Entropy is maximum (with a value of 1) if one ordering is just as probably as the reverse (a sign of weak dependency) and minimum (with a value of 0) if one ordering is much more probable. We negate the entropy value and add a value of 1 to invert its meaning and make it correspond to the required notion of dependency, i.e.:

$$\delta_1(i,j) = 1 - \{-[p_{ij}\log_2(p_{ij}) + (1.0 - p_{ij})\log_2(1.0 - p_{ij})]\} \quad (2)$$

where

$$p_{ij} = \frac{1}{n}\sum_{k=0}^{n-1} \begin{cases} 1 \text{ if } r_i^k < r_j^k \\ 0 \text{ } otherwise \end{cases}$$

and r_i^k is random key i of population member k.

Relative-ordering information is not the only type of information that is potentially of importance in permutation optimization. Consider for instance the traveling salesman problem. The fact that one city is visited before another becomes increasingly important if the *proximity* of the two cities in the permutation is small, to the point where one city is followed directly by the other. This is also known as *adjacency information*. To convey this notion of dependency, we compute the average squared difference between two random keys. This value lies in $[0, 1]$. Because a larger difference indicates weaker dependency, we negate this value and add a value of 1 to invert its meaning and make it correspond to the required notion of dependency, i.e.:

$$\delta_2(i,j) = 1 - \frac{1}{n}\sum_{k=0}^{n-1}(r_i^k - r_j^k)^2 \quad (3)$$

3.3 Random Rescaling

Even if variable dependencies are successfully detected, mixing the associated partial solutions using the GOM operator does not necessarily guarantee that they are combined properly. This is especially the case if (weak) dependencies exist *between* partial solutions, which is quite likely for permutation problems. There may for instance be 5 integers that need to appear in a specific sequence, but together they must also appear before (or after) 5 other integers. Mixing the partial solutions pertaining to the random keys for these integers does not help to ensure this because every random key may have any value in $[0, 1]$. Hence, the larger linkage sets are, the larger the chance that they will end up mixed because the largest random key in the one partial solution is larger than the smallest random key in the other partial solution. For this reason we consider an additional operator, called random rescaling, that increases the chance that partial solutions are combined properly in case dependencies between partial solutions exist. This operator was previously introduced for a model-based EA named $\mathbb{ICE}$ (Iterated density-estimation evolutionary algorithm Induced Chromosome Elements Exchanger) [3]. In $\mathbb{ICE}$, crossover points are identified based on probabilistic modeling. These crossover points are used on random-keys encoded permutations. Without random rescaling, a hard problem based on overlapping sub-functions could not be solved with $\mathbb{ICE}$.

With random rescaling, a block of random keys that is transferred to an offspring is scaled to a randomly selected sub-interval of $[0, 1]$ with probability p_ϱ. For instance, if $(0.1, 0.2, 0.3)$ is scaled to $[0.9, 0.95]$, we get $(0.9, 0.925, 0.95)$. Note that this does not change the sub-permutation that is encoded, thus respecting the relative ordering of the partial solution. We adhere to the settings that led to the best results obtained with $\mathbb{ICE}$ and use $p_\varrho = 0.1$ and ℓ equidistantly sized sub-intervals of $[0, 1]$ for random rescaling.

3.4 Re-encoding

Even with random rescaling, the diversity in the population may still be reduced quickly because the GOM operator combines solutions extensively, especially if the linkage tree is used as the linkage model. Moreover, random rescaling only has an impact if mixing leads to an improvement. Hence, the permutations that can be constructed are severely dependent on the random-key values generated upon initialization. To combat this potential lack of diversity, we consider a second operator that we call re-encoding. With this operator, at the beginning of each generation, the entire population and the overall elitist (i.e. best-ever evaluated, in any population) solution are re-encoded using newly sampled random keys encodings. To re-encode a permutation $\boldsymbol{x}$, first a vector $\boldsymbol{s}$ of ℓ random values $\in [0, 1]$ is generated. Next, the (ascending) sorting orders of both $\boldsymbol{x}$ and $\boldsymbol{s}$ are determined. Denoting these orders by $\sigma(\boldsymbol{x})$ and $\sigma(\boldsymbol{s})$ respectively, the new, random keys encoding $\boldsymbol{r}$ for $\boldsymbol{x}$ is obtained by setting $\boldsymbol{r}_{\sigma(\boldsymbol{x})_i} = \boldsymbol{s}_{\sigma(\boldsymbol{s})_i}$, for all $i \in \{0, 1, \ldots, \ell - 1\}$.

3.5 Selection and Variation

Because of the random keys encoding, in principle no changes are needed for the variation procedure as defined in Section 2.3 to work. However, the Forced Improvement (FI) threshold for the NIS of $\lfloor 1 + \log_{10}(n) \rfloor$ has been established based on benchmark results for Cartesian-space discrete optimization problems. Although forced improvements improve convergence speed, diversity drops quickly. Given that the size of permutation search spaces grows factorially with ℓ, which is faster than the exponential growth of Cartesian spaces, the importance of maintaining diversity in permutation spaces is likely greater. Even more importantly, we proposed to study the added use of random rescaling in Section 3.3 because we argued that mixing blocks of random keys may not be enough to exploit certain types of dependency. Random rescaling however acts much like a mutation operator, which may require more time to be effective than the typical fast convergence of GOMEA allows for.

Partially, the need to maintain diversity longer is addressed by re-encoding (Section 3.4). However, re-encoding only increases genotypic diversity, not phenotypic diversity (i.e. the actual integer permutations). We therefore also consider postponing the FI phase. It is typically during this time that mutation-like operators need some time to have a positive impact. As a first, simple, experiment, we therefore consider the impact of using a FI threshold that requires the NIS to be 10 times longer, i.e.: $10 + 10\lfloor 1 + \log_{10}(n) \rfloor$.

3.6 Population Sizing Scheme

The termination of GOMEA instances with smaller populations is not necessarily beneficial in the case of random keys. For similar reasons as in the case of the FI threshold (Section 3.5), pertaining to diversity maintenance and the

manner in which solutions are combined and improved, more generations may be required to find improvements than in the case of discrete Cartesian search spaces. Therefore, even if the average fitness of a smaller population is overtaken by that of a larger population, there is a good chance that the smaller population may still generate promising solutions. This would mean that it may not be beneficial to terminate smaller populations prematurely. Therefore, we will study the impact of *not* adopting this criterion in our implementation of the population sizing scheme. In that case, a population is only terminated if all diversity is lost, i.e. it has converged to all the same solutions. Because of the FI phase in GOMEA, such convergence happens efficiently.

4. BENCHMARK PROBLEM: PERMUTATION FLOWSHOP SCHEDULING

We consider a well-known permutation-based optimization problem: the permutation flowshop scheduling problem (PFSP). In the following, we present a definition of PFSP, instances used for benchmarking, performance measures to compare EAs with and the allowed computing budget.

4.1 Problem Definition

Solving PFSP involves optimizing a job processing sequence for J jobs on M machines. Each job i requires M uninterrupted operations following the same order $j = 1, 2, \ldots, M$, and operation j can only be processed on machine j. Each machine can process the corresponding operation of only one job at a time. If operation j of job i needs to be processed, but machine j is running another job, then job i will need to wait until machine j finishes its current work. Let $p(i, j)$ be the processing time of job i on machine j. A job-processing sequence $\boldsymbol{\pi}$ is given by $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_J\}$, where π_1 indicates the first job and π_J indicates the last job. The completion time $c(\pi_i, j)$ of the i-th job on machine j, i.e., the duration from the time that the i-th job is started on the first machine until the time that the i-th job is finished on machine j, can be computed as follows (see, e.g. [9]):

$$\begin{aligned}
&c(\pi_1, 1) = p(\pi_1, 1) \\
&c(\pi_1, j) = c(\pi_1, j-1) + p(\pi_1, j), \quad \text{for } j = 2, \ldots, M \\
&c(\pi_i, 1) = c(\pi_{i-1}, 1) + p(\pi_i, 1), \quad \text{for } i = 2, \ldots, J \\
&c(\pi_i, j) = \max\{c(\pi_{i-1}, j), c(\pi_i, j-1)\} + p(\pi_i, j), \\
&\qquad \text{for } i = 2, \ldots, J; \quad j = 2, \ldots, M
\end{aligned} \tag{4}$$

The completion time of the i-th job on the last machine M, i.e. $c(\pi_i, M)$, is called the flow time of the i-th job. The total flow time (TFT) of a sequence $\boldsymbol{\pi}$ can now be computed by summing the flow times of each job:

$$TFT(\boldsymbol{\pi}) = \sum_{i=1}^{J} c(\pi_i, M) \tag{5}$$

The objective that we consider to define PFSP is to find the job-processing schedule $\boldsymbol{\pi}^*$ that minimizes TFT (Equation 5). This is a commonly used objective, but it should be noted that various objectives are used in literature.

4.2 Problem Instances for Benchmarking

A well-known set of PFSP instances, often used for benchmarking purposes, are the so-called Taillard instances [15]. There are 120 such instances, consisting of 10 instances for each of 12 different combinations of J jobs $\times$ M machines: 20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×10, 200×20, and 500×20. The optimal solutions for these instances are unknown. Only the best-known upper bound per instance is available (i.e. the TFT of the best solution ever found). Here, we adhere to the upper bounds as presented in a recent major publication on PFSP targeted at minimizing the TFT [5].

4.3 Performance Measures

For each problem instance in the Taillard benchmark and for each EA, $N = 20$ independent runs are performed, similar to the publication on GM-EDA in which also 20 runs were used [5]. Studies on PFSP often use the average relative percentage deviation (ARPD) to base the comparison of algorithms on:

$$ARPD = \frac{1}{N} \sum_{i=1}^{N} \frac{100(TFT_i - UB)}{UB} \tag{6}$$

where TFT_i is the total flow time of the best solution found in the i-th run and UB is the best-known upper bound of the problem instance being considered. Using the average to compare the performance of algorithms using common statistical significance tests implies that the TFT_i have to be normally distributed, which is typically not true. Therefore, we propose that, in addition to average, the median of the relative percentage deviations (MRPD) should also be investigated as it can be used for the use of statistical hypothesis tests that do not assume normality (see Section 5.2), i.e.:

$$MRPD = \text{median}\left(\bigcup_{i=0}^{n-1} \left\{ \frac{100(TFT_i - UB)}{UB} \right\}\right) \tag{7}$$

4.4 Computing Budget

For fair comparison, each EA should be assigned the same computing budget. Since configurations of computers can vary widely, instead of execution time we use the total number of solution evaluations as the computing budget. Moreover, note that GOMEA could still be sped up substantially, requiring less time or being able to evalute more solutions in the same time, by using partial evaluations (i.e. computing only the change in fitness that results from changing only a subset of variables in GOM). We do not use this possibility here however for fairness and because we approach problems from a black-box perspective. Because we will compare the performance of GOMEA with GM-EDA, we use the same total number of evaluations as reported to have been used in the publication on GM-EDA applied to PFSP [5]. These numbers are reproduced in Table 1.

J **jobs** $\times$ M **machines**	**Maximum evaluations**	J **jobs** $\times$ M **machines**	**Maximum evaluations**
20×5	182224100	100×5	235879800
20×10	224784800	100×10	266211000
20×20	256896400	100×20	283040000
50×5	220712150	200×10	272515500
50×10	256208100	200×20	287728850
50×20	275954150	500×20	260316750

Table 1: Computing budget, taken from [5].

5. RESULTS

In this section we present results of solving 120 Taillard PFSP instances with GOMEA and GM-EDA.

5.1 GOMEA Variants and Configurations

To study the impact of changing the FI threshold and of not prematurely terminating smaller populations (see Sections 3.6 and 3.5) we consider 3 GOMEA *configurations*:

1. The original configuration of GOMEA as defined for Cartesian search spaces.
2. Not prematurely stopping smaller populations, but still using the original FI threshold of $1 + \lfloor^{10}\log(n)\rfloor$.
3. Not prematurely stopping smaller populations as well as using the new FI threshold of $10 + 10\lfloor^{10}\log(n)\rfloor$.

We have furthermore introduced two additional operators that potentially improve performance: random rescaling (X) and re-encoding (R) (see Sections 3.3 and 3.4). To study the impact of these two operators, we performed all experiments with four GOMEA *variants*:

- **O-O**: without re-encoding and without rescaling.
- **O-R**: with re-encoding and without rescaling.
- **X-O**: with rescaling and without re-encoding.
- **X-R**: with rescaling and with re-encoding.

5.2 Comparing Algorithm Outcomes

Outcomes are tabulated in Tables 2 and 3. For each problem instance, the EA that obtains the smallest MRPD value (i.e., the median) is considered to be the best and its results are typeset in boldface. For reference, ARPD values (i.e., the average) are also presented, in parentheses. We have used the Mann-Whitney-Wilcoxon statistical hypothesis test for equality of medians with $p < 0.05$ to see whether the best EA performs statistically significantly better than the other EAs. We select this test because it requires only the assumption that the *shape* of distribution underlying these outcomes is the same and the outcomes here are all the result of running an EA. Although more general pairwise tests exist (e.g., Kolmogorov Smirnov) that do not require this assumption, these tests are also less specific, i.e. weaker. Because we perform multiple pairwise tests, we must account for the increase in probability of making at least one statistical error. Hence, we employ the Bonferroni correction, which implies that the value of p is divided by $m-1$ where m is the number of EAs that the best one is compared with. Results for an instance that are statistically significantly worse than the best result are shaded in gray. Finally, new best values found with GOMEA for a problem instance are underlined.

5.3 Initial Results for Configuration Selection

To find the best GOMEA configuration, we first perform experiments on a subset of ten 50×5 PFSP Taillard instances. We do not use the smaller $20\times M$ Taillard instances because they are quite easy to solve for any GOMEA variant, making it harder to draw any conclusions.

The results presented in Table 2 clearly show that the additional operators of random rescaling and re-encoding have a positive impact on performance. In its Cartesian-space configuration (configuration 1), the obtained solutions of all four GOMEA variants are far from the best-known upper bounds. Moreover, we observed that population sizes up to 2^{19} were initialized when the computing budget was spent (i.e., 220712150 evaluations for 50×5 PFSP instances). This indicates that many smaller populations were quickly terminated because variation did not generate better solutions quickly enough. All GOMEA variants in configuration 1 clearly perform worse than GM-EDA.

However, when the termination criterion of small populations is omitted (i.e., configuration 2), the results of all GOMEA variants substantially improve. Results are comparable to, or even better than, those obtained with GM-EDA. Results are even better for configuration 3. In 9 out of 10 problem instances, the new FI threshold brings about excellent performance, which is also statistically significantly better than virtually all results of GOMEA variants in configurations 1 and 2 (almost all cells for these configurations are gray shaded). These results experimentally confirm our conjecture that enhanced diversity preservation has a positive effect on the performance of the random-keys GOMEA. Based on these initial results, we deem GOMEA in configuration 3 to be the most suitable for a full-benchmark comparison, which we turn to in the next Section.

5.4 Full Benchmark Comparison

The results obtained by all four GOMEA variants on all 120 PSFP Taillard instances are shown in Table 3. For the purpose of comparison, the results of GM-EDA are reproduced here from their original publication [5].

On the 30 smallest problem instances (20×5, 20×10, 20×20), all four GOMEA variants reach the best-known solutions in *every* run (i.e., the ARPD is 0), performing statistically significantly better than the state-of-the-art GM-EDA that did not always find the best-known solution in every run. In order to achieve this, GM-EDA must be combined with variable neighborhood search (VNS), which is a type local search that has been shown to be highly effective for PFSP [5].

In general, GOMEA variants, particularly X-O and X-R, almost always significantly outperform GM-EDA, except for the instances of size 200×10 and 200×20. For the 200×10 instances, while GM-EDA does obtain better results on average, in most cases the difference with the results obtained by various GOMEA variants is not statistically significant.

On the largest problem instances, in the set 500×20, the results obtained by any GOMEA variant is not only statistically significantly better than the results obtained by GM-EDA, but the difference is also substantial. The median/mean distances from the results of GOMEA to the best-known upper bounds are 6 times smaller than those from the results of GM-EDA. Although not shown here, the results of GOMEA variants are even better than the results obtained by the aforementioned hybridized version of GM-EDA with VNS (see [5]). One reason for this may be that the population size of GM-EDA was fixed to $10\times J$ in [5], where J is the number jobs. This may not be the optimal population size. However, this population size was determined with careful consideration. Also, GM-EDA was allowed to restart upon convergence, as long as the computing budget was not used up yet, which reduces somewhat the need for selecting an optimal population size. At the same time, although the population-management scheme that renders all GOMEA variants in this paper free of the population size parameter, presents a big practical advantage, the simultaneous operation of multiple populations of different sizes always induces some degree of inefficiency. Even with this inefficiency, GOMEA still manages to outperform GM-EDA on a majority of the Taillard instances, even when consider-

ing only a single GOMEA variant (e.g., X-R). On a side note, we argue that the inefficiency of the population-management scheme can easily be justified by the great added value in terms of usability, allowing practitioners to use GOMEA without the need for any time-consuming parameter tuning.

While the O-O variant (i.e., without re-encoding and without random rescaling) already works fairly well on all PFSP instances (in configuration 3), even when solving the biggest instances (set 500×20), the results still leave room for improvement. The random-rescaling operator (X-O) appears to be have a bigger positive impact on performance than the re-encoding operator (O-R), especially when the size of instances increases. The combination of both re-encoding and random rescaling (X-R) improves the positive effect of random rescaling for some cases, but it also degrades the performance (of X-O) in several cases. Hence, comparing the performance of the X-O and X-R variants yields alternating results, depending on the problem instance. For the largest instances (set 500×20), the X-R variant appears to be the best, but the obtained results are not statistically significantly different from those obtained by the X-O variant.

Finally, considering the best-known results as reported in the latest major publication on PFSP [5] in which besides GM-EDA, also various local search algorithms and hybrid EAs were tested, our GOMEA variants found new best-known solutions (i.e. new upper bounds) for 2 problem instances in the set 50×5 (underlined in Tables 2 and 3). This further underlines the promise and potential of GOMEA, both for discrete Cartesian and permutation spaces, because it is very difficult for a black-box, general-purpose solver (i.e. without the use of specialized (local search) operators) to obtain solutions of higher quality than those found by state-of-the-art problem-specific algorithms. Given this excellent basis, combining GOMEA with powerful, problem-specific, local search is a worthwhile future extension.

6. CONCLUSIONS

In this paper, for the first time, we proposed a Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) for permutation optimization problems. We did so by making use of the random-keys encoding so that the known strengths of GOMEA for solving discrete Cartesian-space optimization problems can be harnessed effectively. We further employed a population-management scheme that eliminates the need for manually setting a population size and two specific additional operators, random rescaling and population re-encoding, to maintain diversity longer and to increase the chance of effectively processing dependencies *between* partial solutions. Using the linkage tree for the linkage model, the performance of the novel GOMEA for permutation optimization problems was tested on 120 problem instances of the permutation flowshop scheduling problem (PFSP), targeted at minimizing total flow time. Results confirmed that the excellent performance of GOMEA that was previously observed for discrete Cartesian search spaces, is retained. Even though no (problem-specific) local search was employed, GOMEA reliably obtained high-quality solutions, significantly outperforming the recent state-of-the-art generalized Mallows estimation-of-distribution algorithm (GM-EDA), and even finding new best-known solutions for two problem instances. Altogether, the proposed novel GOMEA based on random keys may be considered to be a promising new EA for permutation optimization.

7. ACKNOWLEDGMENTS

We thank Josu Ceberio and Jose A. Lozano for providing us with outcomes of all runs of GM-EDA on the Taillard instances so that we could perform statistical hypothesis tests.

8. REFERENCES

[1] J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994.

[2] P. A. N. Bosman, J. Grahl, and D. Thierens. Benchmarking parameter-free AMaLGaM on functions with and without noise. *Evolutionary Computation*, 21(3):445–469, 2013.

[3] P. A. N. Bosman and D. Thierens. Crossing the road to efficient ideas for permutation problems. In *Proceedings of the Genetic and Evolutionary Computation Conference — GECCO–2001*, pages 219–226, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[4] P. A. N. Bosman and D. Thierens. More concise and robust linkage learning by filtering and combining linkage hierarchies. In *Proceedings of the Genetic and Evolutionary Computation Conference — GECCO–2013*, pages 359–366, New York, New York, 2013. ACM Press.

[5] J. Ceberio, E. Irurozki, A. Mendiburu, and J. Lozano. A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. *IEEE Trans. on Evolutionary Computation*, 18(2):286–300, 2014.

[6] I. Gronau and S. Moran. Optimal implementations of UPGMA and other common clustering algorithms. *Information Processing Letters*, 104(6):205 – 210, 2007.

[7] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA–ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[8] G. R. Harik and F. G. Lobo. A parameter-less genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), 13-17 July 1999, Orlando, Florida, USA*, pages 258–265, 1999.

[9] C.-Y. Hsu, P.-C. Chang, and M.-H. Chen. A linkage mining in block-based evolutionary algorithm for permutation flowshop scheduling problem. *Computers & Industrial Engineering*, 83:159–171, 2015.

[10] S.-H. Hsu and T.-L. Yu. Optimization by pairwise linkage detection, incremental linkage set, and restricted / back mixing: DSMGA-II. In *Proceedings of the Genetic and Evolutionary Computation Conference — GECCO–2015*, pages 519–526, New York, New York, 2015. ACM Press.

[11] F. Lardeux, F. Saubion, and J.-K. Hao. GASAT: A genetic local search algorithm for the satisfiability problem. *Evolutionary Computation*, 14(2):223–253, 2006.

[12] N. H. Luong, H. L. Poutré, and P. A. N. Bosman. Exploiting linkage information and problem-specific knowledge in evolutionary distribution network expansion planning. In *Proceedings of the Genetic and Evolutionary Computation Conference — GECCO–2015*, pages 1231–1238, New York, New York, 2015. ACM Press.

[13] M. Pelikan and D. E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference — GECCO–2001*, pages 511–518, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[14] M. Pelikan, K. Sastry, and E. Cantú-Paz. *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*. Springer–Verlag, Berlin, 2006.

[15] E. Taillard. Project management anf scheduling benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278 – 285, 1993.

[16] D. Thierens. Scalability problems of simple genetic algorithms. *Evolutionary Computation*, 7(4):331–352, 1999.

[17] D. Wierstra, T. Schaul, and T. Glasmachers. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2015.

		Configuration 1				Configuration 2				Configuration 3			
	Best	O-O	O-R	X-R	X-R	O-O	O-R	X-O	X-R	O-O	O-R	X-O	X-R
50 × 5	64803	4.08 (4.11)	4.10 (4.10)	3.79 (3.74)	3.75 (3.74)	0.34 (0.32)	0.22 (0.27)	0.13 (0.17)	0.18 (0.21)	0.11 (0.12)	0.09 (0.09)	0.10 (0.09)	**0.06 (0.07)**
	68062	4.77 (4.76)	4.66 (4.65)	4.15 (4.13)	4.22 (4.15)	0.52 (0.51)	0.46 (0.45)	0.32 (0.31)	0.35 (0.36)	0.27 (0.27)	0.21 (0.19)	**0.16 (0.15)**	0.18 (0.16)
	63162	4.90 (4.89)	4.90 (4.87)	4.80 (4.73)	4.67 (4.65)	1.29 (1.33)	0.91 (0.91)	1.23 (1.22)	1.06 (1.09)	0.83 (0.81)	**0.62 (0.62)**	0.86 (0.87)	0.71 (0.75)
	68226	4.75 (4.73)	4.71 (4.68)	4.47 (4.42)	4.49 (4.46)	1.04 (1.04)	0.88 (0.92)	0.68 (0.76)	0.85 (0.86)	0.85 (0.87)	0.80 (0.79)	**0.25 (0.29)**	0.67 (0.66)
	69392	4.16 (4.15)	4.07 (4.06)	3.96 (3.93)	4.03 (3.94)	0.73 (0.68)	0.58 (0.62)	0.53 (0.59)	0.58 (0.62)	0.44 (0.47)	0.36 (0.37)	0.32 (0.33)	**0.25 (0.26)**
	66841	4.09 (4.08)	4.08 (3.99)	3.81 (3.79)	3.80 (3.82)	0.41 (0.43)	0.44 (0.45)	0.39 (0.45)	0.46 (0.45)	**0.27 (0.23)**	0.28 (0.28)	0.28 (0.25)	0.31 (0.31)
	66253	4.21 (4.19)	4.12 (4.11)	3.97 (3.93)	4.04 (4.00)	0.56 (0.60)	0.43 (0.43)	0.47 (0.49)	0.41 (0.41)	0.25 (0.25)	0.37 (0.36)	**0.12 (0.11)**	0.18 (0.20)
	64359	4.48 (4.39)	4.45 (4.42)	4.04 (4.03)	4.13 (4.08)	0.49 (0.50)	0.29 (0.28)	0.48 (0.47)	0.37 (0.34)	0.31 (0.31)	0.32 (0.29)	0.36 (0.35)	**0.23 (0.22)**
	62981	4.36 (4.36)	4.28 (4.23)	4.03 (4.04)	4.11 (4.07)	0.68 (0.70)	0.48 (0.44)	0.43 (0.47)	0.42 (0.43)	0.31 (0.34)	**0.26 (0.27)**	0.28 (0.31)	0.26 (0.25)
	68853	4.48 (4.41)	4.31 (4.32)	4.32 (4.24)	4.08 (4.06)	0.81 (0.80)	0.73 (0.69)	0.82 (0.84)	0.81 (0.78)	0.49 (0.48)	**0.34 (0.35)**	0.37 (0.40)	0.44 (0.40)

Table 2: Results of GOMEA variants on selected PFSP Taillard instances. For details on formatting, see Section 5.2.

	Best	O-O	O-R	X-O	X-R	GM-EDA		Best	O-O	O-R	X-O	X-R	GM-EDA
20 × 5	14033	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.19 (0.18)	100 × 5	253713	1.02 (0.95)	0.60 (0.62)	0.86 (0.83)	**0.56 (0.58)**	0.83 (0.82)
	15151	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.40 (0.48)		242777	1.20 (1.21)	0.85 (0.88)	**0.51 (0.53)**	0.62 (0.63)	1.00 (1.00)
	13301	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.48 (0.50)		238180	1.51 (1.53)	1.06 (1.10)	**0.59 (0.63)**	0.73 (0.76)	0.78 (0.80)
	15447	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.43 (0.43)		227889	1.46 (1.45)	1.13 (1.11)	**0.45 (0.53)**	0.69 (0.74)	0.78 (0.78)
	13529	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.00 (0.21)		240589	1.19 (1.20)	1.05 (1.05)	**0.58 (0.60)**	0.73 (0.71)	0.82 (0.80)
	13123	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.00 (0.08)		232936	1.83 (1.79)	1.22 (1.24)	1.11 (1.10)	1.04 (1.03)	**0.79 (0.80)**
	13548	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.75 (0.79)		240669	1.43 (1.43)	0.80 (0.78)	**0.59 (0.63)**	0.75 (0.77)	1.01 (1.00)
	13948	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.14 (0.18)		231428	1.92 (1.94)	1.54 (1.54)	**0.59 (0.66)**	0.94 (1.02)	0.88 (0.90)
	14295	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.00 (0.18)		248481	1.59 (1.61)	1.24 (1.16)	0.98 (1.04)	1.12 (1.06)	**0.92 (0.87)**
	12943	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.46 (0.46)		243360	1.44 (1.44)	1.01 (1.01)	**0.65 (0.69)**	0.95 (0.91)	0.92 (0.96)
20 × 10	20911	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.53 (0.45)	100 × 10	299431	2.00 (2.00)	1.88 (1.80)	1.85 (1.83)	**1.61 (1.61)**	1.71 (1.69)
	22440	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.53 (0.54)		274593	1.56 (1.63)	**1.06 (1.22)**	1.38 (1.31)	1.24 (1.17)	2.06 (2.07)
	19833	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.22 (0.31)		288630	1.73 (1.68)	1.46 (1.45)	1.32 (1.25)	**1.06 (1.09)**	1.68 (1.71)
	18710	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.66 (0.75)		302105	1.83 (1.73)	1.52 (1.43)	**0.90 (1.08)**	1.28 (1.27)	1.96 (1.89)
	18641	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.39 (0.35)		285340	2.00 (1.94)	1.78 (1.85)	**1.05 (1.03)**	1.57 (1.62)	1.72 (1.73)
	19245	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.59 (0.77)		270817	**1.31 (1.39)**	1.38 (1.39)	1.35 (1.39)	1.35 (1.31)	1.59 (1.70)
	18363	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.46 (0.48)		280649	1.30 (1.29)	0.81 (0.84)	1.08 (1.07)	**0.70 (0.75)**	1.48 (1.51)
	20241	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.41 (0.47)		291665	1.88 (1.90)	**1.08 (1.12)**	1.57 (1.61)	1.17 (1.10)	1.83 (1.88)
	20330	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.08 (0.27)		302624	1.51 (1.48)	1.19 (1.23)	1.13 (1.22)	**0.93 (0.96)**	1.80 (1.76)
	21320	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.07 (0.24)		292230	1.83 (1.82)	1.32 (1.40)	1.32 (1.22)	**1.31 (1.35)**	1.46 (1.50)
20 × 20	33623	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.57 (0.65)	100 × 20	367267	1.90 (1.91)	1.97 (1.98)	**1.09 (1.08)**	1.56 (1.53)	2.04 (2.03)
	31587	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.22 (0.29)		374032	1.11 (1.18)	**0.75 (0.78)**	0.90 (0.94)	0.82 (0.84)	1.79 (1.80)
	33920	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.00 (0.04)		371417	1.93 (1.88)	1.74 (1.74)	**1.68 (1.70)**	1.82 (1.77)	1.97 (1.93)
	31661	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.24 (0.28)		373822	1.75 (1.86)	1.88 (1.92)	**1.06 (1.12)**	1.92 (1.78)	1.89 (1.86)
	34557	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.33 (0.26)		370459	1.90 (1.94)	**1.04 (1.08)**	1.31 (1.33)	1.10 (1.09)	1.76 (1.77)
	32564	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.10 (0.30)		372768	2.21 (2.12)	2.15 (2.07)	**1.66 (1.65)**	2.14 (2.08)	2.20 (2.17)
	32922	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.58 (0.61)		374483	2.03 (1.92)	1.86 (1.84)	**1.10 (1.10)**	1.58 (1.69)	1.90 (1.90)
	32412	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.58 (0.52)		385456	1.82 (1.86)	1.61 (1.66)	**1.35 (1.41)**	2.04 (1.82)	1.96 (1.96)
	33600	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.33 (0.56)		376063	1.90 (1.84)	1.92 (1.98)	1.89 (1.76)	**1.77 (1.72)**	1.85 (1.82)
	32262	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	**0.00 (0.00)**	0.18 (0.41)		379899	1.94 (1.84)	**1.58 (1.68)**	1.66 (1.72)	1.68 (1.55)	2.09 (2.05)
50 × 5	64803	0.11 (0.12)	0.09 (0.09)	0.10 (0.09)	**0.06 (0.07)**	0.74 (0.79)	200 × 10	1047662	1.97 (1.95)	1.39 (1.37)	1.83 (1.80)	**0.99 (1.07)**	1.16 (1.19)
	68062	0.27 (0.27)	0.21 (0.19)	**0.16 (0.15)**	0.18 (0.16)	0.88 (0.94)		1036042	2.72 (2.70)	2.47 (2.45)	2.07 (1.99)	2.12 (1.95)	**1.47 (1.46)**
	63162	0.83 (0.81)	**0.62 (0.62)**	0.86 (0.87)	0.71 (0.75)	1.32 (1.34)		1047571	2.21 (2.13)	1.80 (1.76)	1.29 (1.33)	1.17 (1.18)	**1.13 (1.12)**
	68226	0.85 (0.87)	0.80 (0.79)	**0.25 (0.29)**	0.67 (0.66)	1.18 (1.27)		1032095	2.06 (2.06)	1.69 (1.59)	1.20 (1.21)	1.24 (1.19)	**1.13 (1.13)**
	69392	0.44 (0.47)	0.36 (0.37)	0.32 (0.33)	**0.25 (0.26)**	0.84 (0.89)		1037053	2.19 (2.18)	2.04 (1.98)	1.42 (1.45)	1.47 (1.44)	**1.31 (1.32)**
	66841	**0.27 (0.23)**	0.28 (0.28)	0.28 (0.25)	0.31 (0.31)	0.80 (0.82)		1006650	2.06 (2.14)	1.71 (1.78)	1.49 (1.47)	**1.26 (1.28)**	1.36 (1.39)
	66253	0.25 (0.25)	0.37 (0.36)	**0.12 (0.11)**	0.18 (0.20)	0.99 (0.96)		1053390	2.15 (2.16)	1.95 (1.94)	1.73 (1.70)	1.29 (1.37)	**1.18 (1.17)**
	64359	0.31 (0.31)	0.32 (0.29)	0.36 (0.35)	**0.23 (0.22)**	0.95 (0.97)		1046246	2.07 (2.02)	1.92 (1.83)	1.39 (1.43)	1.50 (1.43)	**1.25 (1.26)**
	62981	0.31 (0.34)	**0.26 (0.27)**	0.28 (0.31)	0.26 (0.25)	0.84 (0.81)		1025145	1.96 (1.95)	1.77 (1.86)	1.21 (1.36)	1.14 (1.16)	**1.13 (1.11)**
	68853	0.49 (0.48)	**0.34 (0.35)**	0.37 (0.40)	0.44 (0.40)	0.93 (1.00)		1031176	2.40 (2.42)	1.92 (1.88)	1.61 (1.47)	1.45 (1.44)	**1.32 (1.29)**
50 × 10	87207	0.76 (0.84)	0.64 (0.69)	0.75 (0.78)	**0.62 (0.67)**	2.16 (2.10)	200 × 20	1226879	2.05 (2.08)	2.00 (1.92)	1.63 (1.56)	**1.59 (1.62)**	1.60 (1.59)
	82820	0.69 (0.64)	**0.51 (0.56)**	0.63 (0.61)	0.53 (0.50)	2.41 (2.45)		1241811	2.32 (2.26)	2.24 (2.22)	1.55 (1.71)	2.18 (2.12)	**1.46 (1.45)**
	79987	0.48 (0.46)	**0.37 (0.36)**	0.56 (0.56)	0.43 (0.41)	1.78 (1.84)		1266153	2.15 (2.13)	1.72 (1.81)	1.60 (1.59)	1.67 (1.73)	**1.35 (1.33)**
	86581	0.99 (1.00)	0.71 (0.68)	**0.48 (0.47)**	0.58 (0.64)	1.78 (1.83)		1237053	2.38 (2.30)	2.13 (2.24)	1.49 (1.51)	2.04 (2.10)	**1.47 (1.43)**
	86450	0.48 (0.54)	0.52 (0.55)	**0.41 (0.45)**	0.45 (0.47)	2.09 (2.01)		1223551	2.21 (2.14)	2.12 (2.13)	2.27 (2.23)	2.17 (2.14)	**1.61 (1.64)**
	86637	0.44 (0.44)	0.33 (0.34)	0.33 (0.39)	**0.28 (0.29)**	1.48 (1.55)		1225254	2.22 (2.26)	2.29 (2.24)	2.17 (2.32)	2.19 (2.06)	**1.51 (1.52)**
	88866	0.94 (0.89)	0.78 (0.79)	**0.55 (0.57)**	0.57 (0.62)	1.94 (1.97)		1241847	1.98 (1.95)	1.82 (1.80)	1.83 (1.78)	1.67 (1.62)	**1.27 (1.27)**
	86824	0.91 (0.88)	**0.61 (0.63)**	0.62 (0.68)	0.75 (0.69)	1.95 (2.03)		1240820	2.30 (2.36)	1.81 (1.89)	**1.52 (1.62)**	2.00 (1.92)	1.55 (1.57)
	85526	1.00 (1.02)	0.70 (0.69)	0.66 (0.72)	**0.63 (0.67)**	2.09 (2.10)		1229066	2.41 (2.28)	2.27 (2.25)	1.70 (1.74)	1.90 (1.93)	**1.47 (1.48)**
	88077	0.66 (0.62)	**0.52 (0.53)**	0.69 (0.73)	0.65 (0.62)	2.00 (2.00)		1247156	2.14 (2.14)	2.01 (1.97)	1.61 (1.57)	1.85 (1.91)	**1.47 (1.44)**
50 × 20	125831	**0.46 (0.50)**	0.52 (0.54)	0.69 (0.70)	0.54 (0.51)	1.71 (1.76)	500 × 20	6708053	2.11 (2.08)	1.60 (1.60)	1.57 (1.51)	**1.49 (1.49)**	9.11 (8.90)
	119259	0.24 (0.26)	0.16 (0.14)	0.47 (0.54)	**0.15 (0.18)**	1.55 (1.58)		6829668	1.89 (1.83)	1.55 (1.48)	1.56 (1.56)	**1.50 (1.47)**	8.45 (8.58)
	116459	0.71 (0.68)	0.64 (0.61)	**0.58 (0.60)**	0.58 (0.54)	2.17 (2.24)		6747387	2.11 (2.06)	1.68 (1.68)	1.56 (1.47)	**1.49 (1.50)**	8.57 (8.46)
	120712	0.52 (0.54)	**0.44 (0.50)**	0.67 (0.62)	0.53 (0.49)	1.79 (1.92)		6787054	1.91 (1.88)	1.51 (1.50)	1.50 (1.49)	**1.43 (1.46)**	8.78 (8.75)
	118184	0.77 (0.79)	**0.72 (0.73)**	0.99 (0.96)	0.72 (0.73)	2.38 (2.30)		6755257	1.89 (1.92)	1.54 (1.52)	1.55 (1.59)	**1.29 (1.32)**	8.78 (8.72)
	120703	0.65 (0.63)	0.49 (0.57)	0.55 (0.57)	**0.39 (0.40)**	1.81 (1.78)		6751496	2.01 (1.99)	1.64 (1.53)	1.47 (1.53)	**1.41 (1.42)**	8.32 (8.58)
	122962	1.11 (1.06)	0.96 (0.96)	**0.95 (0.92)**	1.05 (1.03)	2.06 (2.10)		6708860	2.08 (2.09)	1.71 (1.67)	**1.63 (1.61)**	1.63 (1.64)	9.25 (9.15)
	122489	0.82 (0.84)	0.70 (0.74)	0.94 (0.97)	**0.69 (0.65)**	2.26 (2.24)		6769821	2.18 (2.17)	1.73 (1.67)	1.72 (1.74)	**1.58 (1.55)**	8.54 (8.62)
	121872	0.48 (0.54)	0.37 (0.50)	**0.36 (0.41)**	0.38 (0.41)	1.78 (1.79)		6720474	1.99 (1.98)	1.53 (1.51)	1.58 (1.67)	**1.43 (1.45)**	8.81 (8.69)
	124064	1.09 (1.08)	1.06 (1.01)	**0.63 (0.63)**	1.07 (1.03)	1.98 (1.95)		6767645	1.97 (2.00)	1.64 (1.63)	1.66 (1.63)	**1.45 (1.46)**	8.60 (8.51)

Table 3: Results of GOMEA variants (in configuration 3) and GM-EDA on all PFSP Taillard instances. For details on formatting, see Section 5.2.