systems

Article

An Estimation of Distribution Algorithm for Permutation Flow-Shop Scheduling Problem

Sami Lemtenneche [1,*], Abdallah Bensayah [2] and Abdelhakim Cheriet [1,3]

[1] LINATI Laboratory, University of Kasdi Merbah Ouargla, BP 511, Ouargla 30000, Algeria; abdelhakim.cheriet@univ-ouargla.dz
[2] Laboratory of Applied Mathematics, University of Kasdi Merbah Ouargla, BP 511, Ouargla 30000, Algeria; bensayah.abdallah@univ-ouargla.dz
[3] RPL Laboratory, University Mohamed Khider, BP 145 RP, Biskra 07000, Algeria
* Correspondence: lemtennech@gmail.com

Abstract: Estimation of distribution algorithms (EDAs) is a subset of evolutionary algorithms widely used in various optimization problems, known for their favorable results. Each generation of EDAs builds a probabilistic model to represent the most promising individuals, and the next generation is created by sampling from this model. The primary challenge in designing such algorithms lies in effectively constructing the probabilistic model. The mutual exclusivity constraint imposes an additional challenge for EDAs to approach permutation-based problems. In this study, we propose a new EDA called Position-Guided Sampling Estimation of Distribution Algorithm (PGS-EDA) specifically designed for permutation-based problems. Unlike conventional approaches, our algorithm focuses on the positions rather than the elements during the sampling phase. We evaluate the performance of our algorithm on the Permutation Flow-shop Scheduling Problem (PFSP). The experiments conducted on various sizes of Taillard instances provide evidence of the effectiveness of our algorithm in addressing the PFSP, particularly for small and medium-sized problems. The comparison results with other EDAs designed to handle permutation problems demonstrate that our PSG-EDA algorithm consistently achieves the lowest Average Relative Percentage Deviation (ARPD) values in 19 out of the 30 instances of sizes 20 and 50 used in the study. These findings validate the superior performance of our algorithm in terms of minimizing the makespan criterion of the PFSP.

Keywords: estimation of distribution algorithms; permutation-based problems; scheduling problems; evolutionary algorithms

check for updates

Citation: Lemtenneche, S.; Bensayah, A.; Cheriet, A. An Estimation of Distribution Algorithm for Permutation Flow-Shop Scheduling Problem. *Systems* **2023**, *11*, 389. https://doi.org/10.3390/systems11080389

Academic Editors: Fernando De la Prieta Pintado and Vladimír Bureš

Received: 30 June 2023
Revised: 18 July 2023
Accepted: 26 July 2023
Published: 31 July 2023

1. Introduction

Estimation of distribution algorithms (EDAs) is a family of evolutionary optimization algorithms that have received increasing attention in recent years. Also known as probabilistic model-building genetic algorithms (PMBGAs) [1–3], EDAs replace the crossover and mutation operators of genetic algorithms (GAs) [4] with a building and sampling of learned probabilistic model. EDAs based on the idea of modeling the problem's distribution in order to guide the search process to explore the search space. To construct the next generation EDAs samples, the built model. The algorithm terminates when a given criterion is reached, such as a limited number of generations, a stable population, or a lack of progress in recent generations. This algorithm has been successfully applied to a wide range of optimization problems, including continuous and combinatorial problems [2,5–7].

Among the combinatorial problems, permutation-based problems have been the focus of significant research efforts [8]. Permutation-based problems are problems in which the solutions are encoded with permutations of a set of elements. The permutation-based problems pose significant challenges for EDAs, due to the large solution space and the mutual exclusivity constraints.

In the literature, many EDAs have been proposed to solve permutation-based problems, and most of these solutions are extensions of conventional EDAs that were developed to address problems relating to discrete or continuous domains [8]. However, several EDAs have been proposed specifically for permutation problems, including the Node Histogram-based Sampling Algorithm (NHBSA) [9], the Edge Histogram-based Sampling Algorithm (EHBSA) [3], the Random Key EDA (RK-EDA) [7], and the Generalized Mallows Model EDA (GM-EDA) [10]. These algorithms have shown promising results on various permutation-based problems, such as the Traveling Salesman Problem (TSP) [11], the Quadratic Assignment Problem (QAP) [12], and the Permutation Flow-shop Scheduling Problem (PFSP) [13]. However, there is still a need for further improvement in the efficiency and quality of solutions found by EDAs for permutation problems.

Based on the general framework, many EDAs are designed to deal with permutation problems. Ceberio et al. in [8] grouped these algorithms into three principal classes. The first class includes a set of EDAs adapted from the algorithms developed first to solve the problems in continuous domains. Bosman et al. suggests a continuous iterated density estimation evolutionary algorithm (IDEAs) [14] to deal with permutation problems. Univariate Marginal Distribution Algorithm UMDAc [15], Mutual Information Maximization for Input Clustering MIMICc [15] and Estimation of Gaussian Networks Algorithm EGNAc [15], adapted from continuous domain by Lozano and Mendiburu to tackle Jop Shop Scheduling Problem. All approaches in this category are based on the Random Keys algorithm [14].

Although the random key strategy can generate a possible permutation of any vector with a real value by ranking method, the major disadvantage of this technique is the redundancy. The ranking of different vectors of real values may provide the same permutation. The second class of EDAs suggested to solve permutation-based problems is those algorithms adopted from the originally designed to solve problems where the solution is represented as a vector of discrete values. To guarantee obtaining a permutation, the sampling phase was modified, usually, the Probabilistic Logic Sampling algorithm is used. The adapted sampling method is based on the idea of setting 0 to the probability of the previous sampled values and normalizing the probabilities of the remainder of the values to a total of 1.

Furthermore, in recent years many algorithms have been proposed especially to deal with permutation-based problems. In [3] Tsutsui et al propose a new algorithm, Edge Histogram-Based Sampling Algorithm (EHBSA). In EHBSA, an edge is a link between two nodes, the model is built by learning the dependencies between edges and represented with an edge histogram matrix. To generate a new individual, the authors propose two methods of sampling, without and with a template, denoted EHBSA/WO and EHBSA/WT, respectively. The first sampling method samples the edge histogram matrix in ordered positions starting with position 0 and a randomly selected node, and then a roulette wheel is constructed for each row to sample a new node in a way similar to Probabilistic Logic Sampling. The sampling with template uses a parent individual chosen from the previous generation at random as a template, then this template is divided into $n > 2$ segments, choose at random one segment to sample it and the other $n - 1$ segments are copied without changes into the offspring. Another histogram-based algorithm introduced in [9] by the same authors is called the Node Histogram-Based Sampling Algorithm (NHBSA). The NHBSA uses a node histogram matrix which models the distribution of nodes over string absolute positions in the selected individuals. The same two sampling techniques used in EHBSA can be applied in NHBSA.

Another family of EDAs is designed to solve permutation problems in particular. This kind of algorithm is relative to probability models for ranking such as [16,17] in which distance-based exponential models like Mallows Model and Generalized Mallows Model are introduced in EDAs. In [18], the Plackett-Luce probability model is also proposed to learn the distribution over permutation space in the EDA framework. The Generalized

Mallows-EDA [17], and PlackettLuce-EDA [18] obtain good results for PFSP and LOP, respectively.

A random key EDA (RK-EDA) proposed in [7] builds a univariate distribution model like UMDAc with predefined variance. To overcome the redundancy issue of classical random key-based EDAs, the generated RK vectors are rescaled and normalized between $[0, 1]$. In RK-EDA, the authors suggest using a cooling rate parameter to control exploration and exploitation. The RK-EDA tested in different permutation problems and they have good results in the PFSP with respect to the makespan in [7], then using Total Flow Time in [19].

Many estimation of distribution algorithms (EDAs) designed for permutation problems focus on sampling nodes rather than positions. However, this approach can be problematic because the relationship between elements and their positions can significantly affect the algorithm's performance. Furthermore, there is no established method for assigning a node to a specific position, which is often done sequentially or randomly. To our knowledge, no existing EDA for permutation problems has not addressed this issue.

The PFSP is one of the permutation-based problems that has gained more attention in the community due to its wide range of applications, including manufacturing, production planning, textiles, and logistics [20,21]. The PFSP involves scheduling a set of jobs on a set of machines to minimize a specific performance measure. The most used objective function in the studies is makespan [22,23], which refers to the total time necessary to perform all of the jobs on all of the machines. The PFSP is a well-studied problem that has been widely used as a benchmark for various optimization algorithms [19,22].

To minimize the makespan of the PFSP, various approaches have been proposed. These include exact algorithms like the branch and bound algorithm [24,25]. However, as the number of jobs and machines increases, solving PFSP optimally using exact methods becomes computationally challenging due to its NP-hard nature. As a result, heuristic and metaheuristic methods are widely used to tackle this challenge. Several heuristic algorithms have been proposed to obtain approximate solutions for PFSP [26–29]. Additionally, metaheuristic optimization techniques have revolutionized combinatorial optimization, leading to a diverse range of strategies for solving PFSP, including genetic algorithms [30,31], simulated annealing [32], estimation of distribution algorithms [7,10,33], and particle swarm optimization [34,35]. In [36], a parallel metaheuristic approach is designed to tackle PFSP.

In this paper, we propose an EDA for solving permutation-based problems and specifically test it on the PFSP. The proposed algorithm (PGS-EDA) uses a novel sampling technique. The novel sampling method is designed in this work to go deeper into the exploitation of the information captured by the model. Furthermore, our sampling operation is guided by a sequence vector (SV) constructed during the modeling phase. Each element is assigned to a specific position based on the presented order in the sequence vector extracted from the model. This guided sampling approach enables the algorithm to exploit the captured information and effectively explore promising areas in the search space. Our proposal uses a dynamic updating mechanism to adapt the distribution model to the search progress, enabling it to balance exploration and exploitation effectively. We show that the proposed algorithm is able to find high-quality solutions efficiently and outperforms existing state-of-the-art algorithms, through a comprehensive experimental study on a set of benchmark instances of the PFSP.

In the remainder of this paper, an introduction to EDAs is given and PFSP is introduced in detail in the next section. In Section 3, we present the proposed algorithm in detail, including its distribution model, updating mechanism, and selection mechanism. Section 4 describes the experimental setup, including the benchmark instances and the performance measures used. Then, we present the experimental results and a detailed analysis of the performance of the proposed algorithm. Finally, we draw conclusions and suggest directions for future research in Section 5.

2. Problem Definition

This section comprehensively overviews two fundamental aspects: EDAs and the PFSP. By delving into these key components, we establish the necessary background to understand the context and significance of our research within the field.

2.1. Estimation of Distribution Algorithms

In 1996, Muhlenbein et al. [1] proposed the estimation of distribution algorithm (EDA), as a new form of evolutionary algorithm (EA), which was further investigated by Larranaga et al. [37] and Pelikan [38]. The EDA is regarded as an adaption of genetic algorithms (GAs) that does not require crossover or mutation operations. Instead, EDAs build an explicit probabilistic model of the selected promising solutions for each generation to learn the variable probability distribution and characteristics. When the probabilistic model is built, the sampling phase constructs the next generation and leads the algorithm to search for promising areas. The algorithm is repeated until a predefined termination criterion is met. The Algorithm 1 introduces the general pseudocode of EDAs.

Algorithm 1: The general pseudo code of EDAs

```
Input:
$Pop_s$: Population size
$Sel_s$: Selection size
Output:
The best solution
1 $P_0 \leftarrow$ Generate $Pop_s$ individuals uniformly at random;
2 $t \leftarrow 0$;
3 while Termination Criteria not met do
4     Select $Sel_s$ promising individuals from $P_t$;
5     Build probabilistic model $M$ using selected individuals;
6     Sample $M$ to generate new individuals;
7     Update the population $P_t$ with new individuals;
8     $t \leftarrow t + 1$;
9 return The best solution in $P_t$;
```

The big challenge to designing an EDA is building the probabilistic model to collect more information from the individuals. In the context of permutation representation, the mutual exclusivity constraint refers to the fact that each element can only appear once in a permutation to assert $P(\sigma : \sigma(i) = \sigma(j))) = 0 \quad \text{for all} \quad i \neq j$ [8,39]. This constraint makes designing effective algorithms for solving permutation problems in modeling and sampling more challenging. For that reason, researchers have taken a deeper step in developing EDAs by adapting them from continuous or discrete domains to designing specific methods, particularly for tackling permutation problems.

2.2. Permutation Flow-Shop Scheduling Problem (PFSP)

Permutation-based problems are a class of optimization problems characterized by the utilization of permutations as solution representations. These problems are proven to be NP-hard, as demonstrated by Garey and Johnson [40]. In such problems, the solution space consists of all possible permutations of a set of elements, denoted as P_n for a problem size of n. Each permfutation p in P_n represents a specific ordering of the elements. Mathematically, a permutation p can be represented as $p = (p_1, p_2, \ldots, p_n)$, where p_i corresponds to the position of element i in the ordering. The complexity of permutation-based problems arises from the factorial growth of the solution space, which results in an exponential number of potential solutions. The significance of permutation-based problems lies in their broad applicability to diverse domains, including logistics, scheduling, routing, and sequencing. The Traveling Salesperson Problem (TSP) and the Flow-shop Scheduling Problem (PFSP) are notable examples within this class of problems.

In the PFSP, a set of n jobs, denoted as $J = \{J_1, J_2, \ldots, J_n\}$, must be processed on m machines, denoted as $M = \{M_1, M_2, \ldots, M_m\}$. Each job J_i consists of a sequence of operations that must be completed in a specific order on the machines.

To standardize the permutation flow-shop scheduling problem (PFSP), several assumptions are commonly made:

- All tasks start processing at time zero.
- Machines operate continuously without interruptions.
- Tasks must follow a predetermined processing sequence.
- Each machine can process only one task at a time, and vice versa.

These assumptions simplify the problem and allow for a focused analysis of scheduling strategies. However, it is important to recognize that real-world scenarios may deviate from these idealized conditions [22]. For example, variants of the PFSP may include considering machine breakdowns, job release dates, or setup times between operations [41]. A recent study has considered the controllable inspection times in two-machine flow-shop robotic cells optimization [42]. These variants introduce additional complexities and require specialized algorithms and techniques for optimization.

Figure 1 depicts an example of a PFSP where five jobs are processed on four machines. Figure 1 showcases the scheduling arrangement, highlighting that the order of jobs remains consistent across all machines. This means that the sequence of jobs is identical for each machine. Furthermore, Figure 1 demonstrates the strict dependency between operations within a job, as each operation must wait for the completion of the preceding operation on the previous machine. Consequently, Figure 1 represents the sequential execution of operations within each job, adhering to the constraints of the PFSP. The X-axis represents the processing time, while the Y-axis represents the machines. The makespan point on the X-axis corresponds to the completion time of the last job (Job 5) on the final machine (Machine 4), signifying the overall time required to complete all jobs in the scheduling problem.

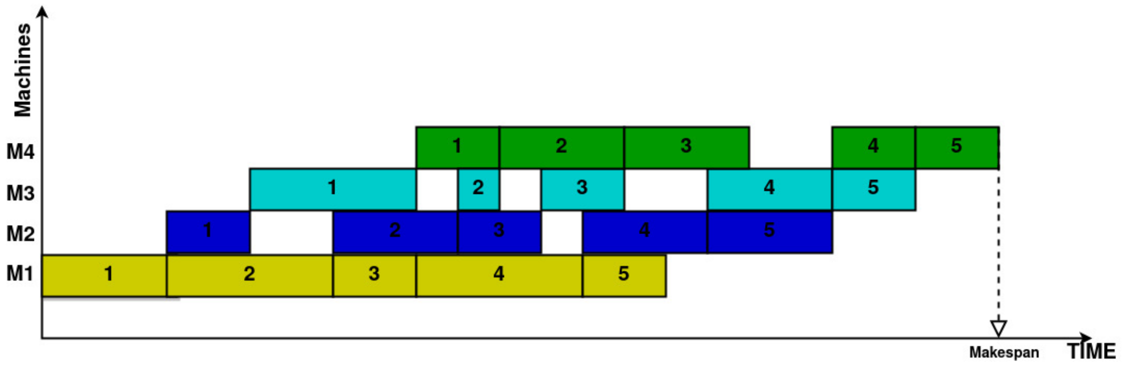

Figure 1. The flow-shop scheduling problem.

The processing time of job J_i on machine M_k is represented by $p_{i,k}$, where $1 \leq i \leq n$ and $1 \leq k \leq m$.

$$C_{\sigma\langle i\rangle,j} = \begin{cases} p_{\sigma\langle i\rangle,j} & i = j = 1 \\ p_{\sigma\langle i\rangle,j} + c_{\sigma\langle i-1\rangle,j} & i > 1, j = 1 \\ p_{\sigma\langle i\rangle,j} + c_{\sigma\langle i\rangle,j-1} & i = 1, j > 1 \\ p_{\sigma\langle i\rangle,j} + \max\{c_{\sigma\langle i-1\rangle,j}, c_{\sigma\langle i\rangle,j-1}\} & i > 1, j > 1 \end{cases} \quad (1)$$

Equation (1) represents the calculation of the completion time $C_{\sigma\langle i\rangle,j}$ for job σ_i on machine j in the PFSP. Let us go through the different scenarios defined by the cases:

- When $i = j = 1$: This corresponds to the first job being scheduled on the first machine. In this case, the completion time is simply the processing time $p_{\sigma\langle 1\rangle,1}$.

- When $i > 1$ and $j = 1$: This represents a job being scheduled on the first machine, but it is not the first job overall. The completion time is the sum of the processing time for the current job $p_{\sigma\langle i\rangle,1}$ and the completion time of the previous job on the same machine $c_{\sigma\langle i-1\rangle,1}$.
- When $i = 1$ and $j > 1$: This corresponds to the first job being scheduled on a machine other than the first one. The completion time is the sum of the processing time for the current job $p_{\sigma\langle 1\rangle,j}$ and the completion time of the previous job on the same machine $c_{\sigma\langle 1\rangle,j-1}$.
- When $i > 1$ and $j > 1$: This represents a job being scheduled on a machine other than the first one, and it is not the first job overall. The completion time is the sum of the processing time for the current job $p_{\sigma\langle i\rangle,j}$ and the maximum value between the completion time of the previous job on the same machine $c_{\sigma\langle i-1\rangle,j}$ and the completion time of the current job on the previous machine $c_{\sigma\langle i\rangle,j-1}$.

The calculation of completion times in the PFSP is obtained regarding all the combinations of job and machine from the previous scenarios.

Thus, the objective of the PFSP is to find the optimal permutation of the jobs that minimizes the makespan, denoted as $C_{\max}$. Equation (2) expresses the makespan as the completion time of the last job n on the last machine m. Equation (3) calculates the makespan as the sum of the processing times of the last operation of the last job on the last machine $P_{n,m}$ and the maximum value between the completion time of the previous job on the same machine $c_{\sigma\langle n-1\rangle,m}$ and the completion time of the current job on the previous machine $c_{\sigma\langle n\rangle,m-1}$. Mathematically, the makespan can be expressed as:

$$C_{\max} = C_{n,m} \tag{2}$$

$$C_{\max} = \max\{C_{n,m-1}, C_{n-1,m}\} + P_{n,m} \tag{3}$$

Algorithm 2 describes the makespan computation for a given permutation and processing time data.

Algorithm 2: Makespan Computing Algorithm

```
   Input: Permutation schedule $\sigma = [\sigma_1, \ldots, \sigma_n]$
   Processing time matrix $P$ of size $n \times m$
   Output: Makespan $C_{\max}$
1  Initialize the completion time matrix $C$ with zeros of size $(n+1) \times (m+1)$;
2  for $j = 1$ to $n$ do
3      for $i = 1$ to $m$ do
4          $C[j,i] \leftarrow Max(C[j-1,i], C[j,i-1]) + P[\sigma_{j-1}, i-1]$;
5  $C_{\max} \leftarrow C[n,m]$;
6  return $C_{\max}$
```

3. The Proposed PGS-EDA Algorithm

In this section, we describe our proposed algorithm in detail. We start with the individual representation and initialization. We then present the new method of modeling and sampling. Finally, we discuss how the population is updated across generations.

The proposed algorithm is based on the standard EDA outline present in Section 2.1. The general steps of PGS-EDA are shown in Algorithm 3.

Algorithm 3: A pseudo code of PGS-EDA

```
Input:
$Pop_s$: Population size
$Sel_s$: Selection size
Output:
The best solution
1  $P_0 \leftarrow$ Generate $Pop_s$ individuals uniformly at random;
2  $t \leftarrow 0$;
3  while Termination Criteria not met do
4      Select $Sel_s$ promising individuals from $P_t$;
5      Build probabilistic model $M$ using selected individuals;
6      From $M$ extract the sequence vector SV;
7      Sampling $Pop_s$ individuals from $M$ according to SV;
8      Update the population $P_t$ with new individuals;
9      $t \leftarrow t + 1$;
10 return The best solution in $P_t$;
```

3.1. Individuals Representation and Initialization

Random initialization strategy is one commonly employed approach for generating the initial population in evolutionary algorithms. To achieve an initial population with a desired level of diversity, the initial population is randomly and uniformly generated [43]. In this work, the individuals are represented by permutations of jobs. Each permutation corresponds to a particular arrangement of jobs. During the scheduling process, the jobs are allocated to the machines in the order specified by the permutation.

3.2. Modeling

At each generation, a set of individuals are selected according to their fitness. Each individual in the population is represented by a permutation σ_i as described in Equation (4).

$$\sigma_i = (\sigma_i(1), \sigma_i(2), \sigma_i(3), \ldots, \sigma_i(n)) \tag{4}$$

where n is the problem size. The proposed algorithm learns a probabilistic model that indicates a particular element's count or frequency at that position across the population. In other words, it describes how often the element i appears in position j.

The model is represented as a matrix M of n elements, as follows:

$$m_{ij} = \sum_{i=1}^{m} \alpha_{ij} + \varepsilon \tag{5}$$

where m in the size of selected individuals, and α_{ij} is a function used to count the appearance of the elements defined as:

$$\alpha_{ij} = 1 \quad if \quad \sigma_i = j, \tag{6}$$

and ε is a user-defined constant used to contribute to the diversification in the sampling phase, and used for numerical stability and avoiding mathematical complications when dealing with zero-valued.

Therefore, we can formulate our model (M) as a matrix of size $n \times n$, as follows:

$$M = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix}$$

Adding epsilon ϵ to all elements of M:

$$M = \begin{bmatrix} a_{11}+\epsilon & a_{12}+\epsilon & \dots & a_{1n}+\epsilon \\ a_{21}+\epsilon & a_{22}+\epsilon & \dots & a_{2n}+\epsilon \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}+\epsilon & a_{n2}+\epsilon & \dots & a_{nn}+\epsilon \end{bmatrix}$$

Once the model is constructed, the extraction of the sequence vector (SV) from matrix M plays a crucial role in guiding the subsequent sampling phase. SV, a vector of size n, represents a permutation of elements. The key characteristic of SV is that its elements are arranged based on the frequency of their occurrence in the same position across all individuals or samples. The element that appears most frequently in a specific position across the selected individuals takes precedence and occupies the corresponding position in SV. This arrangement continues for each position, ensuring that the most common elements in each position are placed at the beginning of the SV.

For example, let $Pop(k)$ be the population at the generation k

$$P(k) = \begin{bmatrix} 0 & 1 & 3 & 2 & 4 & 5 \\ 2 & 1 & 0 & 3 & 3 & 5 \\ 4 & 2 & 5 & 1 & 0 & 3 \\ 5 & 2 & 1 & 4 & 3 & 0 \\ 1 & 0 & 5 & 4 & 3 & 2 \\ 1 & 2 & 4 & 0 & 5 & 4 \end{bmatrix}$$

The matrix $M(k)$, which represents the probabilistic model, is constructed for each generation k according to the process described above. Specifically, $M(k)$ is constructed by considering the distribution of elements across string positions in the population $P(k)$ at that particular generation. The value of $\varepsilon = 0.4$ is added to all elements of M.

$$M(k) = \begin{bmatrix} 1.4 & 1.4 & 1.4 & 1.4 & 1.4 & 1.4 \\ 2.4 & 2.4 & 1.4 & 1.4 & 0.4 & 0.4 \\ 1.4 & 3.4 & 0.4 & 1.4 & 0.4 & 1.4 \\ 0.4 & 0.4 & 1.4 & 1.4 & 3.4 & 1.4 \\ 1.4 & 0.4 & 1.4 & 2.4 & 1.4 & 1.4 \\ 1.4 & 0.4 & 2.4 & 0.4 & 1.4 & 2.4 \end{bmatrix}$$

From the matrix M, we can extract the SV. In this example, the SV will be $SV = (2, 3, 1, 4, 5, 0)$ because the element 2 appears three times in the same position 2. Since the element 0 has less frequency, it occurs just one time in each position.

3.3. Sampling

In contrast to other algorithms like NHBSA, EHBSA, or UMDA, where the sampling of a new individual of size n revolves around selecting the element that will occupy the ith position, starting from the first position and progressing to the last, our proposed algorithm takes a different approach by shifting the focus to the position itself rather than the element. The SV represents the order in which elements should be placed in the offspring and serves as a crucial component in our algorithm. During sampling, we extract elements from the SV and determine the appropriate position for each element. By assigning positions based on the SV, we aim to optimize the arrangement of elements within the individual, see Algorithm 4.

Algorithm 4: The sampling algorithm

```
   Input: M : The matrix model, $SV$: The sequence vector
   Output: $Off$ : The new offspring
1  for element $e$ in $SV$ do
2      $PV \leftarrow$ Normalized M[e] ▷ PV is a probability vector obtained from matrix M ;
3      $p \leftarrow$ Sample position from $PV$ ;
4      $Off[p] \leftarrow e$ ;
5      $M[p] \leftarrow 0$;
6  end
7  return $Off$;
```

- Input: The algorithm takes two inputs, Matrix model M: This represents a matrix containing some information or values. Sequence vector SV: This is a vector containing a sequence of elements.
- Initializing the offspring: The algorithm initializes the offspring variable Off, which represents the new offspring. Initially, it is empty or contains default values.
- Iterating over the sequence vector: The algorithm iterates over each element e in the sequence vector SV.
- Computing the probability vector: For each element e, the algorithm computes the probability vector PV by normalizing the corresponding row of the matrix model M. This step ensures that the values in PV sum up to 1 and represent a valid probability distribution.
- Sampling a position: The algorithm samples a position p from the probability vector PV. This step uses a roulette wheel to select an index from PV based on the probabilities associated with each index.
- Assigning the element to the offspring: The algorithm assigns the value of e to the position p in the offspring Off. This step determines where the element e will be placed in the offspring based on the sampled position.
- Updating the matrix model: The algorithm sets the value at position p in the matrix model M to 0. This step ensures that the same position will not be selected again in future iterations, as the corresponding probability will be 0.
- Repeat the process: The algorithm continues iterating over the remaining elements in the sequence vector SV and performs steps 4 to 7 for each element.
- Output: Once the iteration is complete, the algorithm returns the resulting offspring Off, which contains the elements from the sequence vector SV arranged based on the sampled positions.

In summary, the algorithm probabilistically assigns elements from the sequence vector SV to positions in the offspring Off based on the probabilities computed from the matrix model M. By sampling positions and updating the matrix model, the algorithm ensures that each element is placed in a unique position in the offspring, taking into account the probability distribution derived from M.

3.4. Replacement Strategy

The replacement strategy in EDAs determines how the newly generated offspring are incorporated into the existing population. The proposed replacement strategy suggests that an offspring will be added to the population if its fitness exceeds that of the worst individual in the current generation. This selection process ensures that only individuals with superior fitness levels are included in the population, aiming to improve the overall quality of the population over time. Additionally, the strategy ensures that no duplicate individuals are added, maintaining diversity within the population. This approach strikes a balance between exploration and exploitation, gradually replacing weaker individuals with stronger ones to drive the evolution of the population toward better solutions.

4. Experiments and Results

4.1. Experiments

In order to prove the performance of our proposed algorithm, we carried out a set of experiments using a set of 45 instances of the PFSP from Taillard's benchmark of instances [44]. The chosen instances were divided into nine sets of different size (number of jobs $\times$ number of machines) as follows: $20 \times 5, 20 \times 10, 20 \times 20, 50 \times 5, 50 \times 10, 50 \times 20, 100 \times 5, 100 \times 10, 100 \times 20$. We selected the five (05) first instances for each size.

In this work, we compared our proposed method with a set of state-of-the-art EDAs, including Random-Key estimation of Distribution Algorithm (RK-EDA) [7], Univariate Marginal Distribution (UMDA) [15], Generalized Mallows EDAs (GM-EDA) [45], Node Histogram-Based Sampling Algorithm (NHBSA) [9], and the Edge Histogram-Based Sampling Algorithm (EHBSA) [3]. All the algorithms considered for comparison in our study are pure EDAs without hybridization. We specifically excluded hybrid algorithms such as HGM-EDA [10] from our comparison. Our proposed algorithm (PGS-EDA), UMDA, NHBSA, and EHBSA are implemented in Python. The GM-EDA and RK-EDA codes are available on the GitHub of the authors. The parameters used during the experiments for all algorithms are presented in Table 1. The stopping criterion was set to $1000n^2$ fitness evaluation for all algorithms because it is used in many studies [7,8,16]. In addition, we set experimentally $\varepsilon = 0.002$, and $\frac{n}{10}$ interchange operations between two elements in SV before the sampling phase.

During the comparison of different algorithms, we ensured that the parameters recommended by the respective authors were used. It is worth noting that both methods employed sampling without a template for NHBSA and EHBSA. The parameters of RK-EDA used were those suggested by the authors in [7]. Many variants of EDAs use probability distance-based models for ranking, depending on the distance function that is used. In this comparison, we picked GM-EDAs under Kendall$-\tau$ distance [45,46].

Each algorithm taken into consideration for the experiment was run 10 times for each instance. The average relative percentage deviation (ARPD) was used to measure the performance of the proposed algorithm. $ARPD = \frac{1}{10}\sum_{i=1}^{10}\left(\frac{Found_i - Best}{Best} * 100\right)$, where $Found_i$ is the value returned by the algorithm under experiment in the repetition i, and $Best$ is the best-known value for the used instance.

Table 1. The parameters of the algorithm.

Parameter	Value
Population size	$10n$
Selection size	n
Max number of generation	$100n$
Stopping criterion	$1000n$ FEs
Number of runs	10

4.2. Results and Discussion

To better understand the behavior of the algorithms, we divided the Taillard benchmark instances into three sets: the small instances with 20 jobs, the medium instances with 50 jobs, and the large instances with 100 jobs, all on 5, 10, and 20 machines.

Bayesian Performance Analysis (BPA) [47] was used to conduct a statistical analysis of the results obtained in the experiments. BPA is a statistical technique that can be used to study the uncertainty of the results of an experiment. The first step involves transforming the results from ARDP (Average Relative Deviation Percentage) to rankings for each test instance. Then, samples are drawn from the posterior probability distribution of the weights in the Plackett–Luce model, which is a probability model specifically designed for rankings. These sampled weights are used to calculate the probability of each algorithm winning. This process allows us to assess the relative performance of the algorithms based on their rankings and provides a probabilistic measure of their chances of success.

The ARPD results for each algorithm on the selected instances of 20 jobs are presented in Table 2. The best results values are indicated in bold. Our proposed algorithm achieves the best results in terms of the minimum ARPD value in 11 among 15 of the small instances considered. The second most successful algorithm is GM-EDA, outperforming the other methods in 13.33% of the instances, then the RK-EDA and NHBSA both in 6.33%. The UMDA also has good results compared to the EHBSA. The average value of all 15 instances was calculated for each algorithm. In terms of the average ARPD, PGS-EDA demonstrates its effectiveness by achieving the lowest average value compared to the other algorithms. This indicates that PGS-EDA consistently performs well across different instances and showcases its ability to generate high-quality solutions.

Table 2. The ARPD of 20 jobs and 5, 10, and 20 machines of Taillard's benchmarks.

Instance	PGS-EDA	UMDA	RK-EDA	GM-EDA	NHBSA	EHBSA
20×5_1	1.486	1.502	0.310	**0.000**	1.486	1.267
20×5_2	**0.250**	1.530	0.470	0.412	0.423	0.4415
20×5_3	1.267	3.5426	**0.832**	1.378	1.794	3.977
20×5_4	0.959	1.297	1.067	0.765	**0.699**	2.088
20×5_5	1.116	1.262	0.800	**0.622**	1.116	1.197
20×10_1	**0.982**	2.863	2.243	1.245	1.150	3.893
20×10_2	**1.048**	3.640	1.952	1.326	1.115	4.110
20×10_3	**1.537**	5.404	2.152	1.684	1.744	5.41
20×10_4	**1.328**	2.818	1.494	1.291	1.531	4.208
20×10_5	**1.279**	4.756	1.458	1.797	1.747	5.454
20×20_1	**0.959**	3.854	2.137	1.684	1.406	2.873
20×20_2	**1.083**	3.352	1.623	1.666	1.471	3.200
20×20_3	**0.741**	2.665	1.874	1.818	1.216	3.723
20×20_4	**0.890**	2.744	1.363	1.529	1.174	3.193
20×20_5	**0.816**	2.385	1.379	1.580	1.322	3.963
Mean	1.097	2.907	1.410	1.253	2.984	3.260

Figure 2 illustrates the posterior distribution of the probability of each algorithm being the top one. The empirical probabilities are depicted as black dots. It can be observed that the probability of PGS-EDA being the winning algorithm is the highest, often equal to 0.8. According to the results of the Bayesian analysis, the NHBSA is the second-best algorithm. GM-EDA and RK-EDA follow with probabilities around 0.3.

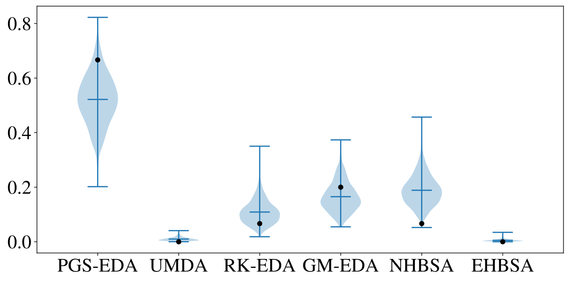

Figure 2. The probability of each algorithm being the best in instances of size 20.

The ARPD results for each algorithm in the selected instances of 50 jobs are presented in Table 3. The best ARPD values are indicated in bold. Our proposed algorithm achieves the best results in terms of the minimum ARPD value in nine of the instances considered. The second most successful algorithm is RK-EDA, outperforming the other methods in four of the instances. The UMDA and NHBSA also have good results compared to the GM-EDA and EHBSA. The makespan of the PFSP is greatly affected by the precise positioning of nodes (jobs). The success of algorithms such as PGS-EDA, RK-EDA, UMDA, and NHBSA can be attributed to their utilization of node-position sampling techniques.

Table 3. The ARPD of 50 jobs and 5, 10, and 20 machines of Taillard's benchmarks.

Instance	PGS-EDA	UMDA	RK-EDA	GM-EDA	NHBSA	EHBSA
50×5_1	**0.000**	0.220	0.0954	0.359	0.055	1.189
50×5_2	0.656	2.183	**0.197**	0.688	0.366	2.194
50×5_3	0.347	0.763	0.148	0.209	**0.076**	1.900
50×5_4	**0.308**	1.768	0.843	0.654	0.367	2.988
50×5_5	**0.034**	0.838	**0.034**	**0.034**	**0.034**	1.847
50×10_1	2.466	3.295	3.246	**1.355**	3.365	8.710
50×10_2	1.943	3.613	1.894	**1.804**	3.692	9.576
50×10_3	2.263	3.690	**1.662**	1.710	3.509	10.391
50×10_4	**1.109**	3.253	1.354	1.174	1.409	9.839
50×10_5	**1.781**	4.377	1.791	1.791	2.106	10.521
50×20_1	**1.762**	3.669	1.809	2.918	3.788	10.699
50×20_2	**2.566**	5.289	3.343	4.209	4.411	11.551
50×20_3	**2.390**	4.929	3.189	3.966	5.697	12.108
50×20_4	1.820	3.587	**1.755**	2.494	3.336	10.621
50×20_5	2.417	4.709	**1.756**	2.825	2.486	9.982
Mean	1.456	3.078	1.539	1.754	2.213	7.607

Notably, our algorithm consistently achieves competitive performance across multiple instances. Our algorithm outperforms other algorithms in 8 out of 15 instances, demonstrating its effectiveness in minimizing the makespan. It is particularly noteworthy that our algorithm achieves the optimal solution in all 10 runs for the first instance of 50 × 5, with ARPD = 0.000, showcasing its superiority in scenarios with a smaller number of machines. On the other hand, the RK-EDA algorithm exhibits strong performance outperforming our algorithm in 4 other instances.

On average, PGS-EDA has the lowest mean ARPD value of 1.456, indicating its overall superior performance across the instances of 50 jobs. GM-EDA, NHBSA, RK-EDA, and UMDA follow with mean ARPD values of 1.754, 2.213, 1.539, and 3.078, respectively. EHBSA has the highest mean ARPD value of 7.607, suggesting that it may struggle to find solutions close to optimality on average.

Figure 3 shows the posterior distribution of the probability of being the top algorithm for each algorithm. The black dots represent the empirical probability for each algorithm to be the best. We can observe that the probability of PGS-EDA being the winning algorithm is the highest, with a probability that can exceed 70%. It has an average probability of around 0.4, followed by RK-EDA and GM-EDA, with probabilities of around 0.3 and 0.2, respectively.

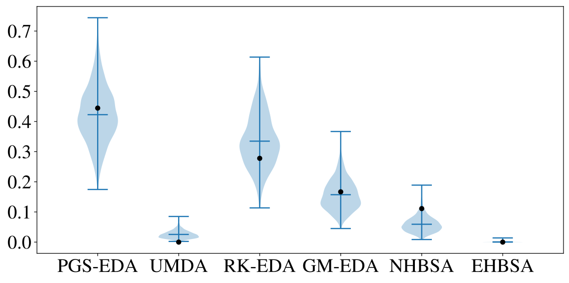

Figure 3. The probability of each algorithm being the best in instances of size 50.

Furthermore, the results of benchmarks on Taillard's instances of 100 jobs are depicted in Table 4. As can be observed, for instances of 5 machines the ARPD values of PSG-EDA, RK-EDA, GM-EDA, and NHBSA are closely the same. The RK-EDA reach a minimum ARPD value in three instances out of five, while the PGS-EDA and NHBSA had one minimum value for both of them. The results indicate that there is no statistically significant difference between the top four algorithms in the small number of machines. This means that the observed difference in performance between those algorithms could reasonably occur by random chance, and there is no strong evidence to support the claim that one algorithm performs better than the other in this specific scenario.

Table 4. The ARPD of 100 jobs and 5, 10, and 20 machines of Taillard's benchmarks.

Instance	PGS-EDA	UMDA	RK-EDA	GM-EDA	NHBSA	EHBSA
100×5_1	0.032	0.190	**0.014**	0.032	0.028	0.709
100×5_2	**0.288**	0.598	0.398	0.335	0.360	1.319
100×5_3	0.206	1.023	**0.102**	0.372	0.502	2.035
100×5_4	0.165	0.597	**0.155**	0.241	0.197	1.804
100×5_5	0.059	0.604	0.057	0.085	**0.047**	2.066
100×10_1	1.018	4.523	**0.519**	1.019	1.057	7.383
100×10_2	1.009	3.047	0.736	**0.646**	1.916	8.104
100×10_3	0.982	5.512	0.246	**0.077**	1.409	6.059
100×10_4	2.469	5.289	**0.694**	1.179	2.538	7.572
100×10_5	1.309	4.322	**0.554**	0.846	1.877	8.247
100×20_1	2.816	7.397	**1.749**	2.083	4.740	11.191
100×20_2	2.371	6.851	**0.903**	1.560	4.718	12.770
100×20_3	4.710	8.523	**0.848**	1.788	4.408	11.376
100×20_4	3.843	7.980	**0.998**	1.665	3.659	11.718
100×20_5	4.861	8.659	**1.655**	2.005	4.496	10.749
Mean	1.742	4.341	**0.641**	0.929	2.130	6.874

In the subset of instances with 10 machines, we observe a slight decrease in the performance of PGS-EDA and NHBSA compared to RK-EDA and GM-EDA. However, it is important to note that PGS-EDA and NHBSA remain competitive in terms of ARPD. Specifically, RK-EDA outperforms the other algorithms in three out of the five instances, demonstrating its effectiveness. Additionally, GM-EDA achieves the minimum ARPD in the remaining two instances, showcasing its strong performance as well. Overall, while there is a variation in performance among the algorithms in this particular subset, PGS-EDA and NHBSA still demonstrate their competitiveness in terms of ARPD. However, when the number of machines increases, the RK-EDA algorithm outperforms all other algorithms and achieves the minimum ARPD among all five instances of 100 × 20. GM-EDA, as the second successful algorithm, outperforms PGS-EDA and NHBSA. Regarding the average value of ARPD, PGS-EDA ranked at position three with 1.471 after RK-EAD and GM-EDA with 0.624 and 0.954, respectively. These findings highlight the importance of considering the specific problem characteristics and the potential impact of varying parameters, such as the number of machines, when comparing and selecting appropriate algorithms for solving permutation-based problems.

The results of the BPA for instances of size 100, which illustrate the probability of each algorithm being the top, are shown in Figure 4. Based on these results, we can conclude that the RK-EDA has the highest probability of being the best algorithm, with an empirical probability of approximately 0.8. Following closely behind are GM-EDA and PSG-EDA, both with a probability of 0.4.

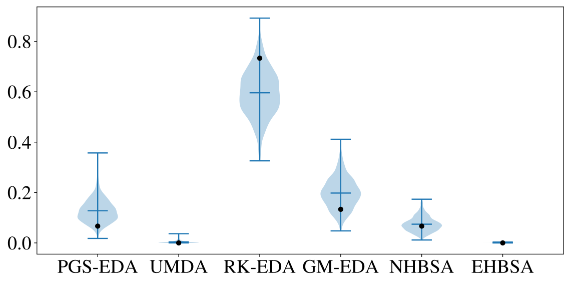

Figure 4. The probability of each algorithm being the best in instances of size 100.

To conclude the statistical analysis, we conducted a Bayesian performance analysis to compare the performance of all the considered algorithms on instances with 20, 50, and 100 jobs. The results, shown in Figure 5, indicate that PGS-EDA has the highest probability of winning, with a probability exceeding 50%. The second-best algorithm is RK-EDA, followed by GM-EDA and NHBSA, respectively. The remaining algorithms have lower probabilities, below 0.05.

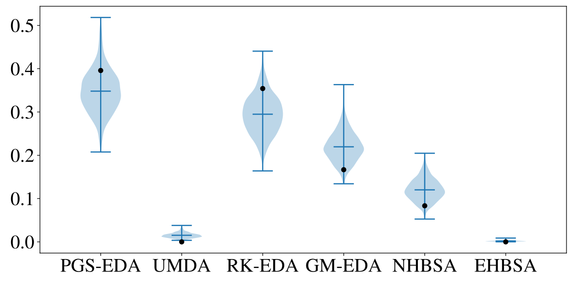

Figure 5. The probability of each algorithm being the best in all instances.

Figure 6 provides insights into the probabilities of each method being among the top two and top three algorithms, as determined by Bayesian performance analysis. The analysis encompassed all instances of size 20, 50, and 100 jobs. The results further validate our previous findings, highlighting the strong performance of PGS-EDA and RK-EDA, which are the most probable algorithms to rank among the top two. This suggests that these algorithms are likely to be the most effective in solving this type of problem. Additionally, when considering the top three algorithms, PGS-EDA maintains its high confidence level, while RK-EDA and GM-EDA closely follow. Notably, NHBSA also exhibits a probability of approximately 0.4 of being among the top three algorithms.

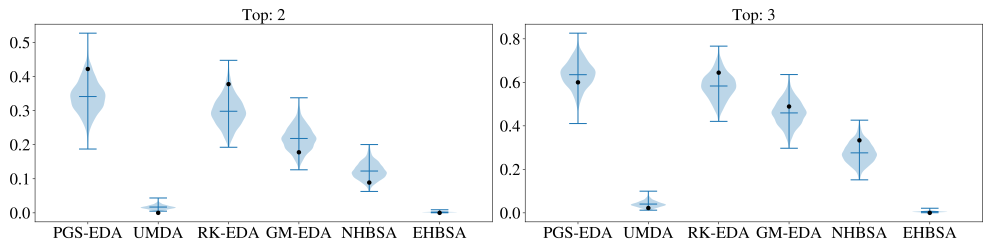

Figure 6. The probability of each algorithm being in the top two and top three in all instances of size 20, 50, and 100.

As a summarization of the statistical results, the Bayesian performance analysis conducted on instances of 20, 50, and 100 jobs reveals significant insights into the relative performance of the algorithms under consideration. PGS-EDA consistently demonstrates the highest probability of being the best algorithm, followed closely by RK-EDA, confirming the authors' suggestion that RK-EDA excels in large-scale problems [19]. GM-EDA and NHBSA also exhibit competitive probabilities, particularly in the top three rankings. However, the remaining algorithms have lower probabilities of being among the top performers. These findings underscore the importance of selecting algorithms based on problem characteristics and varying parameters, highlighting the effectiveness of PGS-EDA and RK-EDA in solving permutation-based problems.

While our algorithm demonstrates promising results on the PFSP, there are limitations to our study that should be acknowledged. The performance of our algorithm may suffer as the problem size increases, and further investigations on a wider range of problem instances would enhance the generalizability of our findings. Additionally, the performance of PGS-EDA may vary when applied to different problem domains or problem sizes.

5. Conclusions

In conclusion, we have introduced a novel estimation of distribution algorithm (EDA) designed specifically for permutation-based problems. Our algorithm incorporates a sampling approach guided by a matrix model and a sequence vector to assign positions to elements in the offspring. Our EDA ensures effective and informed position assignment by leveraging the probabilities derived from the matrix model and the importance ranking of elements in the sequence vector.

We evaluated our EDA on the permutation flow-shop scheduling problem (PFSP) using Taillard's benchmark instances. The results demonstrate the effectiveness of our proposed EDA in addressing the PFSP. Our algorithm consistently achieved competitive performance regarding solution quality, as evidenced by the results on Taillard's benchmark instances. It outperformed or achieved comparable results to state-of-the-art algorithms, showcasing its ability to explore and exploit the solution space effectively.

We suggest several directions for further exploration to address the proposal's limitations and pave the way for future research. Conducting comparative studies with other state-of-the-art algorithms on diverse permutation problems, such as TSP, QAP, or LOP, would provide a more comprehensive benchmarking of our algorithm's performance. Additionally, proposing a hybrid algorithm by integrating local search methods could enhance the performance and scalability of our algorithm.

Author Contributions: Conceptualization, S.L. and A.B.; methodology, S.L. and A.B.; software, S.L. and A.C.; validation, S.L. and A.C.; formal analysis, S.L. and A.C.; resources, A.C.; writing—original draft preparation, S.L.; writing—review and editing, S.L., A.B. and A.C.; visualization, S.L. and A.C.; supervision, A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded in part by the Algerian Ministry of Higher Education and Scientific Research under contract PRFU C00L07UN300120230002.

Data Availability Statement: The data that support the findings of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mühlenbein, H.; Paaß, G. From recombination of genes to the estimation of distributions I. Binary parameters. In *Parallel Problem Solving from Nature, Proceedings of the PPSN IV, Berlin, Germany, 22–26 September 1996;* Voigt, H.M., Ebeling, W., Rechenberg, I., Schwefel, H.P., Eds.; Springer: Berlin/Heidelberg, Germany, 1996; pp. 178–187.
2. Hauschild, M.; Pelikan, M. An introduction and survey of estimation of distribution algorithms. *Swarm Evol. Comput.* **2011**, *1*, 111–128. [CrossRef]
3. Tsutsui, S. Probabilistic Model-Building Genetic Algorithms in Permutation Representation Domain Using Edge Histogram. In *Parallel Problem Solving from Nature, Proceedings of the PPSN VII, Granada, Spain, 7–11 September 2002*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 224–233. [CrossRef]
4. Holland, J.H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*; MIT Press: Cambridge, MA, USA, 1992.
5. Lemtenneche, S.; Cheriet, A.; Abdellah, B. Permutation-Based Optimization Using a Generative Adversarial Network. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '21, Lille, France, 10–14 July 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 159–160. [CrossRef]
6. Cheriet, A. Vine copula-based EDA for dynamic multiobjective optimization. *Evol. Intell.* **2022**, *15*, 455–479. [CrossRef]
7. Ayodele, M.; Mccall, J.; Regnier-Coudert, O. RK-EDA: A Novel Random Key Based Estimation of Distribution Algorithm. In *Parallel Problem Solving from Nature, Proceedings of the PPSN XIV, Edinburgh, UK, 17–21 September 2016*; Springer: Boston, MA, USA, 2016; Volume 9921, pp. 849–858. [CrossRef]
8. Ceberio, J.; Irurozki, E.; Mendiburu, A.; Lozano, J. A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. *Prog. Artif. Intell.* **2012**, *1*, 103–117. [CrossRef]
9. Tsutsui, S. Node Histogram vs. Edge Histogram: A Comparison of Probabilistic Model-Building Genetic Algorithms in Permutation Domains. In Proceedings of the 2006 IEEE International Conference on Evolutionary Computation, Vancouver, BC, Canada, 16–21 July 2006; pp. 1939–1946. [CrossRef]
10. Ceberio, J.; Irurozqui, E.; Mendiburu, A.; Lozano, J. A Distance-Based Ranking Model Estimation of Distribution Algorithm for the Flowshop Scheduling Problem. *IEEE Trans. Evol. Comput.* **2013**, *18*, 286–300. [CrossRef]

11. Goldberg, D.E.; Lingle, R. Alleles, loci, and the traveling salesman problem. In Proceedings of the First International Conference on Genetic Algorithms and Their Applications, Pittsburgh, PA, USA, 24–26 July 1985; pp. 154–159.
12. Koopmans, T.C.; Beckmann, M. Assignment problems and the location of economic activities. *Econom. J. Econom. Soc.* **1957**, *25*, 53–76. [CrossRef]
13. Johnson, S.M. Optimal two- and three-stage production schedules with setup times included. *Nav. Res. Logist. Q.* **1954**, *1*, 61–68.
14. Bosman, P.A.N.; Thierens, D. Crossing the road to efficient IDEAs for permutation problems. In Proceedings of the 2001 Genetic and Evolutionary Computation Conference (GECCO 2001), San Francisco, CA, USA, 7–11 July 2001; pp. 219–226.
15. Lozano, J.A.; Mendiburu, A. Estimation of Distribution Algorithms Applied to the Job Shop Scheduling Problem: Some Preliminary Research. In *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*; Larrañaga, P., Lozano, J.A., Eds.; Springer: Boston, MA, USA, 2002; pp. 231–242. [CrossRef]
16. Ceberio, J.; Mendiburu, A.; Lozano, J. Introducing the Mallows Model on Estimation of Distribution Algorithms. In *Neural Information Processing, Proceedings of the ICONIP 2011, Shanghai, China, 13–17 November 2011*; Springer: Boston, MA, USA, 2011; Volume 7063, pp. 461–470. [CrossRef]
17. Ceberio, J.; Mendiburu, A.; Lozano, J. Kernels of Mallows Models for Solving Permutation-based Problems. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, Madrid, Spain, 11–15 July 2015; pp. 505–512. [CrossRef]
18. Ceberio, J.; Mendiburu, A.; Lozano, J.A. The Plackett-Luce ranking model on permutation-based optimization problems. In Proceedings of the 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 494–501.
19. Ayodele, M.; McCall, J.; Regnier-Coudert, O.; Bowie, L. A Random Key based Estimation of Distribution Algorithm for the Permutation Flowshop Scheduling Problem. In Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC), Donostia, Spain, 5–8 June 2017; pp. 2364–2371. [CrossRef]
20. Liu, H.; Zhao, F.; Wang, L.; Cao, J.; Tang, J.; Jonrinaldi. An estimation of distribution algorithm with multiple intensification strategies for two-stage hybrid flow-shop scheduling problem with sequence-dependent setup time. *Appl. Intell.* **2023**, *53*, 5160–5178. [CrossRef]
21. Sun, L.; Shi, W.; Wang, J.; Mao, H.; Tu, J.; Wang, L. Research on Production Scheduling Technology in Knitting Workshop Based on Improved Genetic Algorithm. *Appl. Sci.* **2023**, *13*, 5701. [CrossRef]
22. Zhang, T.; Guo, X.; Ji, G. Permutation Flow Shop Scheduling Optimization Method Based on Cooperative Games. *IEEE Access* **2023**, *11*, 47377–47389. [CrossRef]
23. Khurshid, B.; Maqsood, S.; Omair, M.; Sarkar, B.; Ahmad, I.; Muhammad, K. An Improved Evolution Strategy Hybridization With Simulated Annealing for Permutation Flow Shop Scheduling Problems. *IEEE Access* **2021**, *9*, 94505–94522. [CrossRef]
24. Brown, A.; Lomnicki, Z. Some applications of the "branch-and-bound" algorithm to the machine scheduling problem. *J. Oper. Res. Soc.* **1966**, *17*, 173–186. [CrossRef]
25. Lomnicki, Z. A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem. *J. Oper. Res. Soc.* **1965**, *16*, 89–100. [CrossRef]
26. Nawaz, M.; Enscore, E.E., Jr.; Ham, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* **1983**, *11*, 91–95. [CrossRef]
27. Koulamas, C. A new constructive heuristic for the flowshop scheduling problem. *Eur. J. Oper. Res.* **1998**, *105*, 66–71. [CrossRef]
28. Semanco, P.; Modrak, V. A comparison of constructive heuristics with the objective of minimizing makespan in the flow-shop scheduling problem. *Acta Polytech. Hung.* **2012**, *9*, 177–190.
29. Kizilay, D.; Tasgetiren, M.F.; Pan, Q.K.; Gao, L. A Variable Block Insertion Heuristic for Solving Permutation Flow Shop Scheduling Problem with Makespan Criterion. *Algorithms* **2019**, *12*, 100. [CrossRef]
30. Caraffa, V.; Ianes, S.; Bagchi, T.P.; Sriskandarajah, C. Minimizing makespan in a blocking flowshop using genetic algorithms. *Int. J. Prod. Econ.* **2001**, *70*, 101–115. [CrossRef]
31. Abdel-Basset, M.; Mohamed, R.; Abouhawwash, M.; Chakrabortty, R.K.; Ryan, M.J. A Simple and Effective Approach for Tackling the Permutation Flow Shop Scheduling Problem. *Mathematics* **2021**, *9*, 270. [CrossRef]
32. Osman, I.H.; Potts, C. Simulated annealing for permutation flow-shop scheduling. *Omega* **1989**, *17*, 551–557. [CrossRef]
33. Pérez-Rodríguez, R. A Hybrid Estimation of Distribution Algorithm for the Quay Crane Scheduling Problem. *Math. Comput. Appl.* **2021**, *26*, 64. [CrossRef]
34. Tasgetiren, M.F.; Liang, Y.C.; Sevkli, M.; Gencyilmaz, G. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *Eur. J. Oper. Res.* **2007**, *177*, 1930–1947. [CrossRef]
35. Hayat, I.; Tariq, A.; Shahzad, W.; Masud, M.; Ahmed, S.; Ali, M.U.; Zafar, A. Hybridization of Particle Swarm Optimization with Variable Neighborhood Search and Simulated Annealing for Improved Handling of the Permutation Flow-Shop Scheduling Problem. *Systems* **2023**, *11*, 221. [CrossRef]
36. Baykasoğlu, A.; Şenol, M.E. Parallel WSAR for Solving Permutation Flow Shop Scheduling Problem. *Comput. Sci. Math. Forum* **2022**, *2*, 10. [CrossRef]
37. Larranaga, P.; Lozano, J.A. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*; Springer Sciencc+Business Media: New York, NY, USA, 2002; p. 398. [CrossRef]
38. Pelikan, M; Goldberg, D.E.; Lobo, F.G. A survey of optimization by building and using probabilistic models. *Comput. Optim. Appl.* **2002**, *21*, 5–20. [CrossRef]

39. Huang, J.; Guestrin, C.; Guibas, L. Fourier Theoretic Probabilistic Inference over Permutations. *J. Mach. Learn. Res.* **2009**, *10*, 997–1070.
40. Garey, M.; Johnson, D. *Computers and Intractability: A Guide to the Theory of NP-Completeness*; Mathematical Sciences Series; W. H. Freeman: New York, NY, USA, 1979.
41. Chandramouli, A.B. Heuristic Approach for N-Job, 3-Machine Flow Shop Scheduling Problem Involving Transportation Time, Break Down Time and Weights of Jobs. *Math. Comput. Appl.* **2005**, *10*, 301–305. [CrossRef]
42. Foumani, M.; Razeghi, A.; Smith-Miles, K. Stochastic optimization of two-machine flow shop robotic cells with controllable inspection times: From theory toward practice. *Robot. Comput.-Integr. Manuf.* **2020**, *61*, 101822. [CrossRef]
43. Jarboui, B.; Eddaly, M.; Siarry, P. An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. *Comput. Oper. Res.* **2009**, *36*, 2638–2646. [CrossRef]
44. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285.
45. Ceberio, J.; Irurozki, E.; Mendiburu, A.; Lozano, J.A. A Review of Distances for the Mallows and Generalized Mallows Estimation of Distribution Algorithms. *Comput. Optim. Appl.* **2015**, *62*, 545–564. [CrossRef]
46. Irurozki, E.; Ceberio, J.; Santamaria, J.; Santana, R.; Mendiburu, A. Algorithm 989: Perm mateda: A Matlab Toolbox of Estimation of Distribution Algorithms for Permutation-Based Combinatorial Optimization Problems. *ACM Trans. Math. Softw.* **2018**, *44*, 1–13. [CrossRef]
47. Rojas-Delgado, J.; Ceberio, J.; Calvo, B.; Lozano, J.A. Bayesian Performance Analysis for Algorithm Ranking Comparison. *IEEE Trans. Evol. Comput.* **2022**, *26*, 1281–1292. [CrossRef]