PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY KASDI-MERBAH OUARGLA

Faculty of New Technologies of Information and Communication

Department of Computer Science and Information Technologies

Thesis Submitted to the Department of Computer Science and Information Technology in Candidacy for the Degree of "Doctor" 3rd Cycle LMD in Computer Science

Option: Artificial Intelligence

Thesis subject:

Model-Based Evolutionary Algorithms and Generative Deep Learning Models for Permutation-Based Problems

Presented by: Sami LEMTENNECHE

Publicly defended on: 07/03/2024 before the jury composed of:

Ahmed KORICHI	Prof	University of Ouargla	President
Abdallah BENSAYAH	Prof	University of Ouargla	Supervisor
Abdelhakim CHERIET	MCA	The National School of Artificial Intelligence	Co-Supervisor
Farouq ZITOUNI	MCA	University of Ouargla	Examiner
Abdelkamel BENALI	MCA	University of El-Oued	Examiner
Brahim LEJDEL	Prof	University of El-Oued	Examiner

Academic Year: 2023/2024

بسم الله الرحمن الرحيم

I dedicate this work, with heartfelt love, to the persons who have been a constant source of inspiration, courage, and hope in my life, my dear parents.

To my lovely wife, my son Mohammed, and daughters Fatima Zahra, Chifa, Asma, and Batoul.

To all my dear brothers and sisters.

To all my family and my friends.

To all those who have provided me with any form of support.

Acknowledgments

I am immensely grateful to my supervisors, **Prof. BENSAYAH Abdallah** and **Dr. CHERIET Abdelhakim**, for their mentorship, encouragement, and unwavering support in shaping this work.

Furthermore, I would like to express my appreciation and thanks to the committee jury members, **Prof. KORICHI Ahmed, Dr. ZITOUNI Farouq, Prof. LAJDEL Brahim, and Dr. BENALI Abdelkamel,** for their constructive feedback and valuable suggestions, which have greatly enriched the quality of this work.

Lastly, I thank my colleagues BELILA Djilani, Dr. BEN BEZZIANE Mohamed, and Dr. MESSIAID Abdessalam for their collaboration and encouragement, making this journey more rewarding and enjoyable.

Contents

List of Figures

List of Tables

Abbreviation List

EDAs Estimation of Distribution Algorithms

PFSP Permutation Flow-Shop Scheduling Problem

GANs Generative AAdversarial Networks

VAEs Variational Auto Encoders

CNN Convolutional Neural Network

RNN Recurrent Neural Networks

EAs Evolutionary Algorithms

GAs Genetic Algorithms

MBEAs Model Based Evolutionary Algorithms

TSP Travelling Salesman Problem

LOP Linear Ordering Problem

QAP Quadratic Assignment Problem

UMDA Univariate Marginal Distribution Algorithm

RK Random Key

RK-EDA Random Key Estimation of Distribution Algorithm

MM Mallows Model

GM-EDA Generalized Mallows Estimation of Distribution Algorithm

NHBSA Node Histogram Based Sampling Algorithm

EHBSA Edge Histogram Based Sampling Algorithm

MIMIC Mutual Information Maximizing Input Clustering

BPA Bayesian Performance Analysis

PGS-EDA Position Guided Sampling Estimation of Distribution Algorithm

VNS Variable Neighbourhood Search

PSO Particle Swarm Optimization

TS Tabu Search

SA Simulated Annealing

PBIL Population-Based Incremental Learning

cGA compact Genetic Algorithm

BMDA Bivariate Marginal Distribution Algorithm

ECGA Extended Compact Genetic Algorithm

DMSGA Dependency-Structure Matrix Genetic Algorithm

IDEA Iterated Density Estimation Evolutionary Algorithm

dtEDA dependency-tree EDA

SV Sequence Vector

Abstract

Estimation of Distribution Algorithms (EDAs) are a type of evolutionary algorithm that are good at solving optimization problems. EDAs work by building a model of the best solutions found so far and then using this model to generate new solutions. The main challenge with EDAs is building a good model. This is especially hard for permutation problems.

We introduce a novel Estimation of Distribution Algorithm (EDA), the Position-Guided Sampling EDA (PGS-EDA), tailored for permutation problems. PGS-EDA focuses on the positions of elements in the solution rather than the elements themselves. This makes PGS-EDA better at solving permutation problems. We tested PGS-EDA on the Permutation Flow-Shop Scheduling Problem (PFSP). Our results showed that PGS-EDA is good at solving the PFSP, especially for small and medium-sized problems. PGS-EDA outperformed other EDAs designed for permutation problems on the PFSP, achieving the lowest Average Relative Percentage Deviation (ARPD).

We also explored the use of Generative Adversarial Networks (GANs) in EDAs. GANs are good at generating samples that look like the training data. However, they have not been well-studied for permutation problems. To address this gap, we proposed a new EDA that uses GANs to estimate the probabilistic model. We represent candidate solutions with one-hot matrices to preserve important information during GAN training. We tested our proposed algorithm on two permutation problems: the Traveling Salesman Problem (TSP) and the PFSP. Our results showed that the algorithm can find the best solution in some cases and near-best solutions in others

Keywords: Evolutionary Algorithms, Estimation of Distribution Algorithms; TSP; Permutation-based problems; Scheduling Problems; Deep Generative Models.

Résumé

Les algorithmes d’estimation de distribution (AED) sont un type d’algorithme évolutionnaire qui est efficace pour résoudre des problèmes d’optimisation. Les EDAs fonctionnent en construisant un modèle des meilleures solutions trouvées, puis en utilisant ce modèle pour générer de nouvelles solutions. Le principal défi des EDAs est de construire un bon modèle. Cela est particulièrement difficile pour les problèmes de permutation. Nous proposons un nouvel EDA appelé Algorithme d’Estimation de Distribution avec Échantillonnage Guidé par la Position (PGS-EDA) qui est spécialement conçu pour les problèmes de permutation. PGS-EDA se concentre sur les positions des éléments dans la solution, plutôt que sur les éléments eux-mêmes. Cela rend PGS-EDA plus efficace pour résoudre les problèmes de permutation. Nous avons testé PGS-EDA sur le problème d’ordonnancement des ateliers en flux de permutation (PFSP). Nos résultats ont montré que PGS-EDA est efficace pour résoudre le PFSP, en particulier pour les problèmes de petite et moyenne taille. PGS-EDA a surpassé les autres EDAs conçus pour les problèmes de permutation sur le PFSP, en obtenant les valeurs d’écart relatif moyen en pourcentage (ARPD). Nous avons également exploré l’utilisation des réseaux adversaires génératifs (GANs) dans les EDAs. Les GANs sont efficaces pour générer des échantillons qui ressemblent aux données d’entraînement. Cependant, ils n’ont pas été bien étudiés pour les problèmes de permutation. Pour combler cette lacune, nous avons proposé un nouvel EDA qui utilise les GANs pour estimer le modèle probabiliste. Nous représentons les solutions candidates par des matrices à chaud afin de préserver les informations importantes pendant l’entraînement du GAN. Nous avons testé notre algorithme proposé sur deux problèmes de permutation : le problème du voyageur de commerce (TSP) et le PFSP. Nos résultats ont montré que l’algorithme peut trouver la meilleure solution dans certains cas et des solutions proches de la meilleure dans d’autres.

Mots-clés: Algorithmes d’Estimation de la Distribution ; Problèmes basés sur les permutations ; Problèmes d’ordonnancement ; Algorithmes évolutionnaires ; Modèles génératifs profonds.

ملخص

خوارزميات تقدير التوزيع هي نوع من الخوارزميات التطورية التي أثبتت نجاعتها في حل العديد مشكلات التحسين. تعمل هذه الخوارزمبات عن طريق بناء نموذج احتمالي لأفضل الحلول التي تم العثور عليها حتى الآن ومن ثم استخدام هذا النموذج لتوليد حلول جديدة. التحدي الرئيسي في تصميم خوارزميات توزبع التقدير هو بناء نموذج جيد. هذا صعب بشكل خاص لمشكلات التبديل.

نقترح في هذا العمل خوارزمية توزبع التقدير جديدة يسمى خوارزمية تقدير التوزيع بتوجيه الموضع والتي تم تصميمها خصيصًا لمشكلات التبديل. تركز هذه الخوارزمية على مواضع العناصر في الحل)التبديلة(، بدلاً من العناصر نفسها. هذا يجعل خوارزمية تقدير التوزيع بتوجيه الموضع المقترحة أفضل في حل مشكلات التبديل. قمنا باختبار هذه الخوارزمية على مشكلة الجدولة التباديلية للتدفق في المعمل . أظهرت نتائجنا أن الخوارزمية المقترحة جيدة في حل هذا النوع من مشاكل التحسين، خاصة بالنسبة للمشكلات الصغيرة والمتوسطة الحجم. حيث تفوقت على خوارزميات تقدير التوزيع أخرى مصممة لمشكلات التبديل في ، حيث حققت أدنى انحراف نسبي نسبي متوسط.

استكشفنا أيضًا استخدام الشبكات التوليدية العميقة خاصة شبكات الخصومة التوليدية في خوارزميات تقدير التوزيع. حيث أن شبكات الخصومة التوليدية جيدة في توليد عينات تشبه بيانات التدريب. ومع ذلك، لم يتم دراستها بشكل جيد لمشكلات التبديل. لمعالجة هذه الفجوة، اقترحنا خوارزمية تقدير التوزيع جديدة يستخدم شبكات الخصومة التوليدية لتقدير النموذج التوزيعي الاحتمالي. حيث نمثل حلول المرشح مع مصفوفات للحفاظ على المعلومات المهمة أثناء تدريب . قمنا باختبار الخوارزمية المقترحة لدينا على مشكلتين من مشكلات التبديل: مشكلة البائع المتجول ومشكلة الجدولة التباديلية للتدفق في المعمل أظهرت نتائجنا أن الخوارزمية يمكنها العثور على أفضل حل ممكن في بعض الحالات وحلول شبه مثالية في حالات أخرى.

الكلمات المفتاحية: خوارزميات تقدير التوزيع، المشاكل القائمة على التبديل، مشاكل الجدولة، الخوارزميات التطورية، النماذج التوليدية العميقة.

Chapter 1

General Introduction

This chapter provides an overview of the scope of this thesis, including the research questions and a summary of the research publications produced during this research.

1.1 Context

Optimization in computer science refers to the procedure of determining the most ideal solution(s) by considering particular criteria. Since many years ago, the study of optimization has attracted a lot of interest among academics, largely because of the critical role it plays in a variety of fields, including industrial, social, financial, and economic activities.

Combinatorial optimization is a branch of mathematical optimization that finds the best solution from a limited set. Note that these potential solutions are discrete, meaning they can only be represented as a graph, a vector, or a permutation. This thesis focuses on Problems in which the solution is expressed as a permutation.

Permutation problems are very difficult to solve when the permutation size is large. In other words, the search space of permutation problems is the set of all possible permutations of n items, so if no constraint is assumed, the number of possible solutions is $n!$. This is because the search space of solutions is exponential in n, meaning that the number of possible permutations grows very quickly as n increases. For example, when $n = 10$, the search space of solutions is 3628800 permutations, and when $n = 20$, it is more than 2×10^{18} permutations. The work of Garey and Johnson [2] showed that many permutation problems are NP-hard, meaning that they are computationally intractable for all but very small instances.

Due to the nature and complexity of permutation problems, the exact methods are not practicable. The limitations of the exact methods encourage the operations research community to suggest many metaheuristics to deal with these problems. Metaheuristics including Tabu search (TS), Variable Neighbourhood search (VNS), Particle Swarm Optimization (PSO), Genetic Algorithm (GA), Estimation of Distribution Algorithms (EDA), Simulated Annealing (SA), and Ant Colony Optimization (ACO) have proven their capacity to provide good solutions in a reasonable time.

Model-based Evolutionary Algorithms (MBEAs) are a class of Evolutionary Algorithms (EAs) that use machine learning models in the evolution process [3]. In this thesis, we are interested in a type of MBEAs called Estimation of distribution Algorithms (EDAs), which use machine learning techniques to build a probabilistic model of selected solutions and then use this model to generate the next generation.

1.2 Motivation and Problematic

Lately, there has been increasing enthusiasm for employing various deep learning models across diverse domains [4]. This interest reflects a recognition of the transformative potential of these advanced machine learning techniques and a growing desire to leverage their capabilities in addressing complex and diverse problems.

Integrating deep neural networks, particularly generative models, into Estimation of Distribution Algorithms (EDAs) is a topic of notable interest within the context of optimization and algorithmic strategies. This fusion marks a convergence between classical optimization methods and cutting-edge machine learning techniques, holding the promise of reshaping the landscape of optimization itself.

GANs and other deep generative models can generate new data, such as images, text, and music. They can also solve problems that involve reordering or rearranging data, such as sequencing tasks and scheduling problems.

This passage is saying that generative models have the potential to change the way we solve permutation problems. Traditional optimization methods for permutation problems can be complex and inefficient. GANs and VAEs may offer a new approach to solving these problems that is more efficient and flexible.

This Work focuses on answering the following research questions:

- How does the predominant node-sampling focus in estimation of distribution algorithms (EDAs), particularly for permutation problems, impact algorithm performance concerning the intricate element-position relationship?
- How does arbitrary node-to-position assignment, in the absence of a well-established method, affect EDA solutions for permutation problems?
- In what way deep generative deep learning models can play the role of learning and sampling probabilistic models in EDAs?
- How can we exploit the capacity of deep generative models to estimate the distribution in permutation space?

1.3 Objectives and Contributions

The primary objective of this study is to leverage the capabilities of estimation of distribution algorithms (EDAs) and generative adversarial networks (GANs) to tackle the complex challenges presented by permutation-based problems. These problems, characterized by the arrangement of elements in sequences or arrangements, pose unique difficulties due to their combinatorial nature and the need to optimize the arrangement of elements according to specific criteria. In this context, the main contributions of this thesis are:

- We provide a comprehensive overview of permutation-based problems, detailing their key characteristics and common variants. By systematically categorizing these problems, we lay the groundwork for understanding their complexity and exploring potential solution approaches.
- Drawing inspiration from the success of GANs in continuous domains, we propose an innovative fusion of GANs and EDAs tailored to permutation problems. This approach

offers a novel perspective on probabilistic model building and sampling, enabling the investigation of GANs' ability to learn and represent the intricate distributions inherent in permutation spaces.

- We undertake a systematic exploration of various permutation representations during GANs training, aiming to evaluate their efficacy in capturing the underlying structure of permutation-based problems. By experimenting with different encoding representation formats, we seek to uncover insights into the effect of the representation on the algorithm performance.
- In response to the specific challenges posed by permutation-based problems, we introduce the Position Guided Sampling Estimation of Distribution Algorithm (PGS-EDA). This novel EDA adopts a position-guided sampling strategy, shifting the focus from individual nodes to positions within permutations during the sampling phase. By prioritizing positional information, PGS-EDA offers a targeted approach to addressing the unique constraints and objectives of permutation optimization.

1.4 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2 is divided into three parts. The first part (Section 2.2) introduces permutation-based problems, a class of challenging optimization problems with real-world applications. This section discusses the unique characteristics of each problem within this domain, providing valuable insights into their complexities.

The second part (Section 2.3) delves into the Estimation of Distribution Algorithms (EDAs), a type of model-based evolutionary algorithm that is particularly well-suited for solving permutation problems. This section provides a comprehensive taxonomy of EDAs, highlighting their diverse aspects. It also focuses on EDAs that have been specifically fine-tuned for permutation problems, setting the stage for understanding novel approaches to addressing these optimization challenges.

The closing section (Section 3) explores the realm of generative deep learning models. It begins with a broad overview of generative models, unraveling their architectural foundations.

This section then delves into the intricate workings of two prominent generative models, Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), elucidating their structural nuances.

Chapter 3 introduces GAN-EDA, a novel EDA-based GAN specifically designed for solving permutation problems. GAN-EDA leverages the power of Generative Adversarial Networks (GANs) to learn a latent probabilistic model of the selected individuals, which is then used to generate new individuals. GAN-EDAs also introduce an alternative permutation representation using one-hot matrices. Additionally, this chapter proposes a hybrid variant of GAN-EDA that incorporates a 2-opt local search algorithm. To evaluate the performance of our proposed algorithm, comprehensive experiments were conducted on benchmark problems such as the Traveling Salesman Problem (TSP) and Permutation Flow Shop Scheduling Problem (PFSP).

Chapter 4 introduces PGS-EDA, a pure EDA-based algorithm designed and tested for solving the Permutation Flow Shop Scheduling Problem (PFSP). The key innovation of PGS-EDA is a new sampling method focusing on sampling positions rather than elements to generate new individuals (permutations). Compared to state-of-the-art algorithms, this new approach has been shown to improve the performance of PGS-EDA on the PFSP, especially for small and medium instance sizes.

In **chapter 5**, as a conclusion, we summarize the key findings of our research, discuss the overall conclusions drawn from our results, and outline potential avenues for future exploration.

Chapter 2

Background and Literature Review

2.1 Introduction

Permutation problems form a subset of combinatorial optimization, where the arrangement of elements matters significantly. These problems hold immense importance due to their relevance in solving real-world challenges across various domains. From optimizing delivery routes and scheduling manufacturing processes to arranging genetic sequences and data points, permutation problems reflect the complexities of decision-making in a wide array of applications. The importance of permutation problems lies in their intricacy, demanding innovative approaches and creative solutions.

This chapter reviews three interconnected domains that are essential to the research presented in this thesis: permutation problems, Estimation of Distribution Algorithms (EDAs). We begin by exploring permutation problems, their real-world applications, their characteristics, and their relevance to optimization. Next, we delve into EDAs, an evolutionary algorithm that has proven effective in solving different optimization problems. This comprehensive review provides the foundation for the research presented in this thesis, which aims to advance our understanding of optimization and problem-solving by drawing insights from these domains.

2.2 Permutation Problems

People frequently utilize permutations in real life due to their flexibility and utility in representing various needs. Permutations can be depicted in numerous distinct forms, such as sorted sets of elements, collections of separate cycles, transpositions, matrices, and graphs. This multitude of representation options results in their appearance across various problems. For example, the assignment of n tasks to n workers, where each worker takes only one task, can be interpreted by a permutation.

Within the realm of combinatorial optimization, the concept of a ”permutation” pertains to a specific set of n natural numbers, which typically range from 1 to n. The key characteristic of these permutations is that each number is uniquely and mutually exclusive, ensuring that no number within the set repeats. This exclusivity plays a fundamental role in the definition and constraints of permutation-based problems. Formally, a permutation of a set S can be defined as a bijective (one-to-one and onto) function that maps each element from the set S to another element within the same set S. In the context of this dissertation, the set S is composed of n items labeled with consecutive natural numbers 1, 2, ..., n. Permutations are specifically represented using the conventional one-line notation, which involves depicting a permutation as an ordered list derived from the set 1, 2, ..., n.Permutations are typically represented by symbols such as σ, π, or τ, where $\sigma(i) = j$ signifies that element j occupies position i. [5]. To prevent any potential confusion in this dissertation, we use the terms permutation, solution, or individual for the same meaning.

Permutation problems belong to a category of optimization problems that employ permutations as a means of representing solutions. These problems have been established as NP-hard, as illustrated by Gareyte and Johnson [2]. The search space in permutation problems, often denoted as S_n, contains all permutations of size n. So, in general, a permutation-based problem consists of finding the best solution σ^* to optimize the fitness function f, in which:

$$f(\sigma^*) < f(\sigma) \quad \forall \sigma \in S_n \tag{2.1}$$

Permutation-based problems encompass a wide range of challenges in which all possible permutations of a set of elements define the solution space. This set is often denoted as (P_n)

when dealing with a problem size of (n). Each permutation σ within (P_n) represents a distinct arrangement or ordering of the elements.

The factorial expansion of the solution space reveals the intrinsic difficulty of permutation-based issues. These issues become computationally difficult due to the exponential number of possible solutions to investigate. The importance of permutation-based problems extends across various fields, such as logistics, routing, and scheduling. They provide versatile models for addressing real-world scenarios with intricate decision-making requirements.

The literature includes a wide range of permutation problems from different fields. In this section, we will introduce four well-known problems: the traveling salesman problem (TSP) [6], the permutation flow-shop scheduling problem (PFSP) [7], the quadratic assignment problem (QAP) [8], and the linear ordering problem (LOP). All of these problems are combinatorial optimization problems whose solutions can be represented as permutations.

2.2.1 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) involves finding the shortest route to visit a set of n distinct cities exactly once and then returning to the starting city. In this problem, a solution is represented as a permutation σ of the cities, where $\sigma(i) = j$ indicates that the $i - th$ stage of the tour visits city j.

For example, in a TSP with four cities, a solution is represented by the permutation σ=3,2,4,1. This tour departs from City 3, then proceeds to City 2, followed by City 4, and finally City 1 before returning to City 3.

The problem is formally described in the following manner: The objective function f is defined in 2.2 as the sum of the distances between all pairs of cities in a given distance matrix $D = [d_{i,j}]_{n\times n}$. The cities are traversed in the order determined by the permutation σ.

$$f(\sigma) = \sum_{i=2}^{n} d_{\sigma(i-1),\sigma(i)} + d_{\sigma(n),\sigma(1)} \tag{2.2}$$

Given our assumption that the initial city in the tour is not predetermined, the Traveling Salesman Problem (TSP) exhibits symmetrical solutions. This symmetry arises because every tour can be depicted using $2n$ distinct permutations in symmetric scenarios, when $d_{i,j} = d_{j,i}$ for all $i, j \in \{1, 2, ..., n\}$. In the asymmetric scenarios, every tour is represented by n distinct permutations. For example, solution $\sigma_1 = \{1, 3, 2, 4\}$ represents the same tour that $\sigma_2 = \{4, 2, 3, 1\}$, because $f(\sigma_1) = f(\sigma_2)$.

From the previous example, we can observe that the absolute position of elements in the permutation is not affected in the fitness calculation. Additionally, if indices i and j are adjacent in the permutation, they contribute equally to the objective function, regardless of their absolute positions. It is important to note that each entry in the distance matrix D is defined by two cities. In summary, we can conclude that the impact of an item $\sigma(i)$ on the fitness function f is directly influenced by its neighboring items: the previous item $\sigma(i-1)$ and the next item $\sigma(i+1)$.

2.2.2 Permutation Flow-Shop Scheduling Problem (PFSP)

In the context of the Permutation Flow-Shop Scheduling Problem (PFSP), we are given n jobs, $J = \{J_1, J_2, \ldots, J_n\}$, and m machines, $M = \{M_1, M_2, \ldots, M_m\}$. Each job, J_i, consists of a sequence of operations that must be processed on the machines in a specific order. The goal of the PFSP is to find a permutation of the jobs that minimizes a specific objective function, such as the makespan (the completion time of the last job on the last machine).

The PFSP is a challenging problem because there is a large number of possible permutations of the jobs, and the completion time of each job depends on the order in which the jobs are processed on each machine.

The PFSP is a complex problem, but it is also a very important problem. It has a wide range of applications in real-world manufacturing and service industries[9, 10]. By standardizing the problem and making some simplifying assumptions, it is possible to develop efficient algorithms for solving the PFSP and finding good solutions to real-world problems.

To standardize PFSP, several standard assumptions are typically applied:

1. **Simultaneous Start:** All tasks commence processing at time zero synchronously.
2. **Uninterrupted Machine Operation:** Machines operate continuously without any interruptions during the scheduling process.
3. **Sequential Processing:** Tasks must adhere to a predefined processing sequence, following a specific order.

4. **Exclusive Machine Task Handling:** Each machine is capable of processing only one task at any given moment, and vice versa. This exclusivity ensures that a machine is dedicated to a single task during its processing.

These assumptions are not always realistic, but they are necessary to make the PFSP tractable to solve. In real-world problems, there may be delays in the start of tasks, machines may break down, and tasks may be interrupted [11]. It is important to note that variants of PFSP can introduce additional complexities beyond the standard assumptions. For instance, variations may include factors such as job release dates, machine breakdowns, or setup times between operations [9]. In a recent study, the consideration of controllable inspection times in the optimization of two-machine flow-shop robotic cells was explored [12].

Figure 2.4 provides an illustrative example of the Permutation Flow-Shop Scheduling Problem featuring 5 jobs executed on 4 machines. This visual representation underscores the consistent job order across all machines. In the context of the PFSP, it is imperative that the job sequence remains consistent across all machines. Additionally, Figure 2.4 vividly illustrates the stringent interdependency between operations within a job, where each operation must await the completion of the preceding operation on the preceding machine. As a result, Figure 2.4 effectively captures the sequential execution of operations within each job, in full compliance with the fundamental constraints of the PFSP. The X-axis is dedicated to representing processing time. The Y-axis corresponds to the machines involved in the scheduling process. Finally, the makespan point on the X-axis denotes the completion time of job 5 (last job) on machine 4 (last machine), symbolizing the total time required to successfully process all jobs within the permutation.

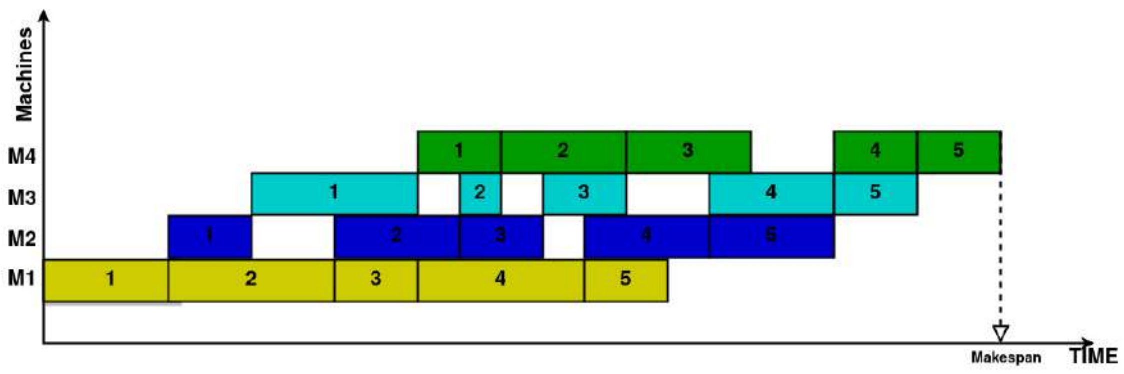

Figure 2.1: An example of PFSP.

$$C_{\sigma\langle i\rangle,j} = \begin{cases} p_{\sigma\langle i\rangle,j} & i=j=1 \\ p_{\sigma\langle i\rangle,j} + c_{\sigma\langle i-1\rangle,j} & i>1, j=1 \\ p_{\sigma\langle i\rangle,j} + c_{\sigma\langle i\rangle,j-1} & i=1, j>1 \\ p_{\sigma\langle i\rangle,j} + \max\{c_{\sigma\langle i-1\rangle,j}, c_{\sigma\langle i\rangle,j-1}\} & i>1, j>1 \end{cases} \tag{2.3}$$

Equation 2.3 provides the formula for calculating the completion time $C_{\sigma\langle i\rangle,j}$ for job σ_i on machine j within the PFSP. The time required to process job J_i on machine M_k is denoted as $p_{i,k}$, with $1 \leq i \leq n$ and $1 \leq k \leq m$. 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 determination of completion times in the Permutation Flow Shop Problem (PFSP) is achieved by considering all possible combinations of jobs and machines from the preceding scenarios.

The influence of element $\sigma(i)$ on the fitness function f is computed based on the cumulative completion times of all tasks scheduled prior to it, ranging from task 1 to task $i-1$, in addition to the processing time of task i. Consequently, the contribution of element i to the fitness function is intricately linked to the specific order in which the preceding $i-1$ tasks are scheduled. This interdependency highlights the critical role of sequencing in determining the overall fitness of the solution.

2.2.3 Quadratic Assignment Problem

The QAP[8] is the problem of assigning a set of *n* facilities to a set of *n* locations, with a cost function calculated according to the distance between locations and flow between facilities. The solution of this problem is codified as a permutation σ of length *n*, where $\sigma_i = j$ represents the facility j is allocated to location j, for example, the solution 2,3,1,4, means that facility 2 is assigned to location 1 and facility 3 is assigned to location 2, and so on. Typically, there are two matrices *n*n* matrices $H = [h_{ij}]$ and $D = [d_{ij}]$ represent respectively the flow between facilities and distance between locations. The objective function can be defined as follows:

$$F(\sigma) = \sum_{i=1}^{n} \sum_{j=1}^{n} h_{ij} \times d_{\sigma(i)\sigma(j)} \tag{2.4}$$

The analysis of the fitness function shows that all possible pairings of items and their positions in the permutation are important for calculating the fitness value. As a result, the impact of item (i) depends on the precise arrangement of the other items in the solution. The solution quality is evaluated based on the absolute position of each facility. Relative to the absolute position of the other indices. As a consequence, the quadratic assignment problem (QAP) is unique among the problems presented because it creates complex relationships between its items.

2.2.4 Linear Ordering Problem

In this permutation problem, we are given a square matrix M of size $n \times n$. The objective is to find a simultaneous permutation of rows and columns of M, where the sum of the super-diagonal entries is maximized, so the solution is a permutation of length n, and the objective function is formulated as follows:

$$F(\sigma) = \sum_{i=1}^{n} \sum_{i=j}^{n} M_{\sigma_i \sigma_j} \tag{2.5}$$

It is clear from this problem that an index's influence on the objective function depends on the indices that come before and after it. In other words, the contribution of the element $\sigma(i)$ is related to the previous elements $(\sigma(1), \ldots, \sigma(i-1)$ and the posterior elements $(\sigma(i+1), \ldots, \sigma(n)$. However, this influence is unaffected by how these previous and following elements are specifically arranged.

2.3 Evolutionary Algorithms

Evolutionary algorithms are population-based metaheuristics for optimization problems. The basic idea behind EAs is inspired by natural evolution and uses their mechanisms, such as selection, mutation, and crossover. This evolution across time leads to the survival of the fittest individuals. To simulate this idea in computation, the community exprime the species as a population of individuals; each individual represents a possible solution to the problem. The traditional example of the EAs is the Genetic Algorithm (GAs) [13, 14], and they also include the evolutionary programming (EP), the evolution strategy (ES), and the genetic programming (GP). Regardless of the different techniques used in different EAs, most of them share the same framework as given in figure 2.2.

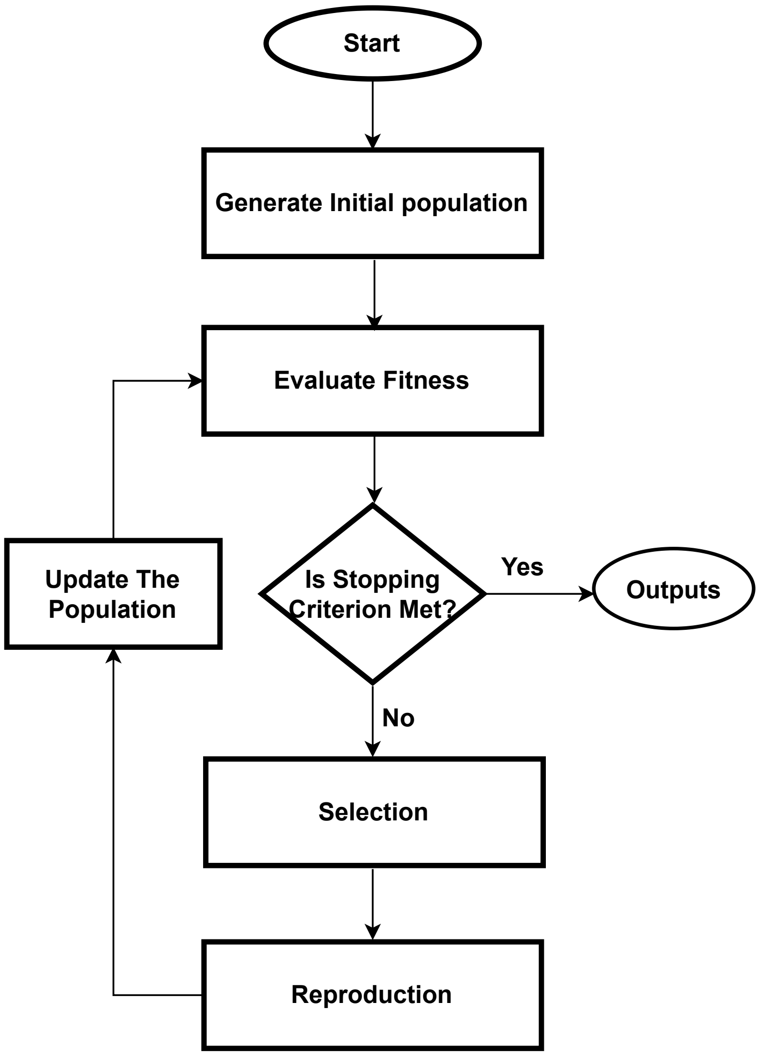

Figure 2.2: The general evolutionary algorithms flow.

The most cited example of EAs is Genetic Algorithms (GAs) [15]. In the reproduction phase, GAs use a crossover (recombination) of two individuals (parents) to combine their genes. Then a mutation operation is applied with a certain probability to the genes of some offspring, see figure 2.3.

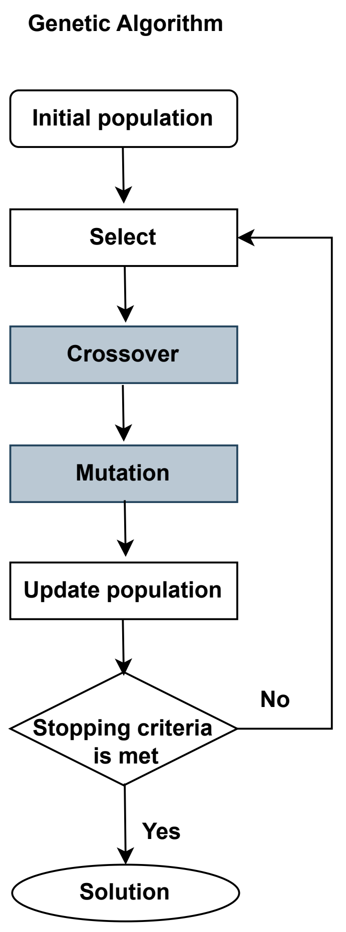

Figure 2.3: A Flow chart for Genetic Algorithm.

- The initial population is a set of solutions to the problem, usually generated at random.
- The fitness of each individual is evaluated. This is a measure of how good the solution is. There are many different ways to evaluate the fitness of an individual, depending on the problem being solved. The algorithm repeats until a termination criterion is met. This could be a maximum number of generations, a certain level of fitness, or a change in the population.
- Individuals with a good fitness score are selected for reproduction. This is usually done

using a selection operator, such as roulette wheel selection or tournament selection [14].

- The selected individuals produce offspring using crossover and mutation operators. Crossover operators combine the genes of two parents to create a new offspring. Mutation operators randomly change the genes of an individual.
- The offspring are evaluated and their fitness is determined.
- The least fit individuals in the population are replaced with the offspring. This ensures that the population always contains a set of good solutions.
- The algorithm returns the individual with the highest fitness. This is the best solution found by the algorithm.

2.4 Estimation of Distribution Algorithms

In 1996, a pioneering work by Muhlenbein et al [16] introduced the Estimation of Distribution Algorithm (EDAs) as a novel evolutionary algorithm (EA), sparking further investigations by researchers like Larranaga et al. [17] and Pelikan[18]. EDAs can be seen as a variation of genetic algorithms (GAs) that eliminates the need for crossover and mutation operations. In fact, EDAs create a probabilistic model for the chosen fittest solutions in each generation, enabling them to learn variable probability distributions. Once the probabilistic model is established, EDAs use this model to sample new individuals and create the next population. This process allows for guiding the algorithm in the search for promising areas. This process is iterated until a predetermined stopping criterion is satisfied. The general pseudocode for EDAs is introduced in Algorithm 1.

Algorithm 1: EDAs pseudo code

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

Estimation of distribution algorithms (EDAs) are a type of evolutionary algorithm that builds a probabilistic model of the search space. This model is used to guide the search for the optimum solution. The model is built by analyzing the features of the selected solutions. Multiple solutions can share these features, and they can represent patterns of interactions between the problem variables [15]. The model can then be used to sample new solutions that are likely to be good.

EDAs hold a distinct advantage over other evolutionary algorithms that do not employ probabilistic models. Frequently, these algorithms struggle to address problems in which significant interactions exist among the problem's variables. EDAs can capture these interactions by building a probabilistic model of the search space [19]. This makes them more robust and scalable than other evolutionary algorithms.

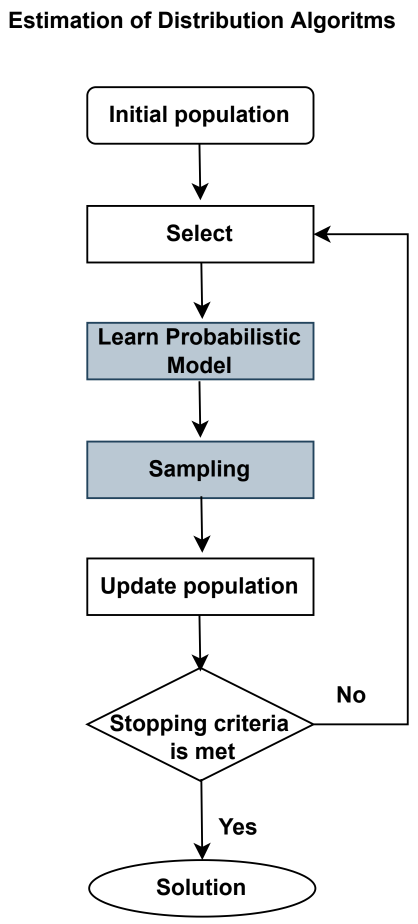

Figure 2.4: A Flow chart for EDAs.

2.4.1 A hands-on illustrative example of EDAs

In order to gain a deeper comprehension of EDAs process, we provide a manual simulation of EDAs. The intent behind this example is to demonstrate the fundamental elements of the basic EDA procedure and to foster a better understanding of how an EDA iteration progresses.

In this example, a simple EDA is used to solve OneMax problems. The OneMax problem is a simple binary optimization problem where the goal is to find a binary string of maximum length that consists of all ones. Each candidate solution (individual) is represented by a binary vector of fixed length n, where $n > 0$. The objective function can be formulated as follows:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i \tag{2.6}$$

In this instance, we employ a population size of $N = 6$, and $n = 5$ binary variables per solution. Truncation selection using a threshold $\tau = 50\%$ is adopted to choose the subset of the most promising solutions (i.e., the top 50% solutions). To model the probability distribution of these promising solutions, a probability vector is used, storing the likelihood of a '1' in each position of the solution strings. This probability vector offers a swift and efficient method for addressing the onemax problem and similar optimization challenges. It capitalizes on the assumption of variable independence. To learn this probability vector, p_i, representing the probability of a '1' at position i, is defined as the proportion of selected solutions containing a '1' in that position. When generating a new binary string from this vector, a '1' is generated at position i with a probability p_i. For instance, if $p_3 = 0.6$, a '1' is introduced at the third position of a new candidate solution with a 60% likelihood. In each generation, $N = 6$ candidate solutions are generated from the current model, shaping a new population of $N = 6$. The process is depicted in figure 2.5.

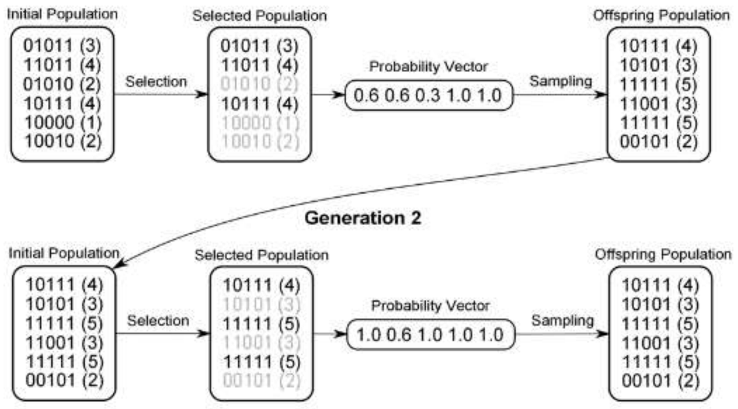

Figure 2.5: EDAs example[1].

Positive results are evident from the first generation onward in this process. Compared to the original population, the number of '1's in the new offspring population has increased significantly, and multiple instances of the global optimal '11111' are present. In addition, the probability of seeing a '1' in any given position has increased, thereby increasing the probability of producing the global optimum. As we progress to the second generation, the probability vector becomes even more pronounced in favor of the global optimal solution. If the simulation were to continue for one more generation, the probability vector would produce the global optimal solution exclusively.

The above example demonstrates the most basic form of EDAs, which assumes a fixed probabilistic model represented by a probability vector [18]. This makes it easy to learn about the model, as there are no other models to choose from. However, this type of EDA is limited in its capabilities.

There are other types of EDAs that allow for more complex probabilistic models that can capture interactions between variables in a problem. These interactions can be learned automatically on a case-by-case basis. This makes the modeling process more complex, but it can be very valuable for solving more complex optimization problems.

2.4.2 EDAs taxonomy

Choosing and building the best probabilistic model for a specific optimization problem can be challenging, as there are many different EDAs available with a variety of models and learning algorithms. One way to make this selection is to consider the trade-off between the complexity of the probabilistic model and the computational cost of storing and learning the model. The complexity of the model is related to the number of variables in the problem [15, 3]. A more complex model can capture more complex relationships between the variables, but it will also be more computationally expensive to store and learn. The computational cost of storing and learning the model is also related to the type of representation used for the variables. A discrete representation, such as binary variables, is typically easier to store and learn than a continuous representation, such as real-valued variables. Ultimately, the best EDA for a specific problem will depend on the particular characteristics of the problem, such as the number of variables, the type of representation, and the desired accuracy.

In this section, we classify EDAs based on the complexity of probabilistic models used to represent the interaction between the variables of the problem.

Univariate EDAs

In order to assess how candidate solutions are distributed within the decision space, considering variable correlation is a crucial aspect of the modeling process. One straightforward approach involves utilizing univariate models, which assume that the decision variables are independent of each other. Under this assumption, the probability distribution of a candidate solution, denoted as $x = (x1, x2, ..., xn)$, can be broken down into the product of individual variable probability distributions:

$$p(x) = p(x_1) * p(x_2) * ... * p(x_n) \tag{2.7}$$

Here, $p(x)$ represents the probability distribution of the candidate solution x, and $p(x_i)$ signifies the probability distribution of the ith decision variable, denoted as xi. Such univariate distribution models are employed in Evolutionary Distribution Algorithms (EDAs) and are referred to as univariate EDAs.

Univariate Evolutionary Distribution Algorithms (EDAs) such as UMDA [16], Population-Based Incremental Learning (PBIL) [20], compact Genetic Algorithm (cGA) [21] are efficient due to their separate univariate modeling. However, their performance can decline when significant interactions between decision variables exist. Such interactions challenge the assumption of variable independence, impacting the accuracy of these algorithms [3, 1]. This limitation has led to the development of more complex multivariate EDAs, which aim to capture interdependencies among variables for better optimization outcomes.

Bivariate EDAs

Bivariate EDAs represent a set of EDAs when the models are learned to capture pair-wise dependencies between variables. Equation 2.8 presents the conditional probability of this models

$$p(X) = p(X_{i_1}|X_{i_2})p(X_{i_2}|X_{i_3}) \cdots p(X_{i_{n-1}}|X_{i_n})p(X_{i_n}) \tag{2.8}$$

Here, $p(X)$ represents the probability distribution of the candidate solution X. Additionally, $p(X_{i_j}|X_{i_{j+1}})$ corresponds to the conditional probability distribution of variable X_{i_j} given variable $X_{i_{j+1}}$, while $i_1, i_2, ..., i_n$ signifies a permutation of the decision variables. This model is also called *tree-based model*; in these models, the probability of a variable is limited to depending on at most one other variable, namely its parent in the tree structure [1].

If each parent has exactly one child, the resulting tree model consists of a single chain of dependencies, which is a special type called *chaine model*. The chain model is used in The Mutual Information Maximizing Input Clustering (MIMIC) [22]. The dependency tree-based EDAs [23] uses a tree model that assumes each parent can possess more than one child. The Bivariate Marginal Distribution Algorithm (BMDA) [24] uses a set of disconnected trees as a graphical model, which refers to the *forest model*.

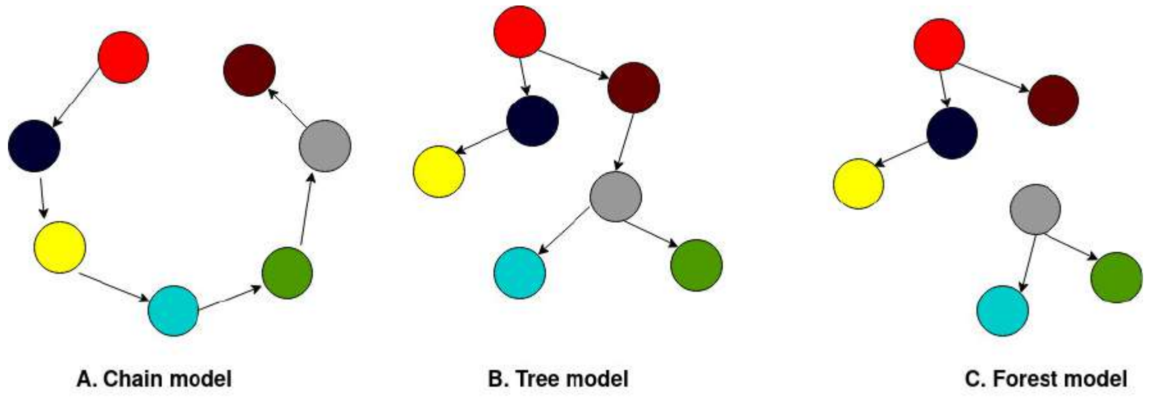

Figure 2.6: Examples of tree-based models

Figure 2.6 illustrates examples of tree-based models, where each variable is represented by a circle, and edges depict the dependencies between variables.

Multivariate EDAs

The conditional probability model in the previous section only considers pairwise interactions between variables. However, it is important to note that some problems may involve more complex interactions between the decision variables. This can limit the model's ability to capture these complexities. Additionally, the complexity of the probabilistic structure increases as the number of dependencies between variables increases. This, in turn, increases the computational effort required to find the most suitable structure for the given data points.

Multivariate dependencies are modeled through either directed acyclic graphs or undirected graphs within multivariate models. In EDAs, two prominent models are used to represent this model, stand out:

- Bayesian networks, which are depicted as directed acyclic graphs where each node represents a variable, and each edge signifies a direct conditional dependence, figure 2.7.A. Equation 2.9 formulate the joint probability distribution of the Bayesian network.

$$P(X) = \prod_{i=1}^{n} P(X_i|parents(X_i)) \tag{2.9}$$

  Where $parents(X_i)$ represents the set of variables for which there exists an edge leading into X_i, and $P(X_i|parents(X_i))$ denotes the conditional probability of X_i given $parents(X_i)$. Bayesian network-based EDAs includes [25, 26]

- Markov networks, which encompass undirected graphs for representing dependencies among variables. However, it is important to note that sampling Markov networks presents a greater challenge compared to sampling Bayesian networks. Put differently, when using Evolutionary Algorithms (EDAs) based on Bayesian networks, some of the complexity shifts from the learning phase to the sampling phase of the probabilistic model. figure 2.7.B. Suggested algorithms in [27, 28, 29] are examples of EDAs based on Markov network.

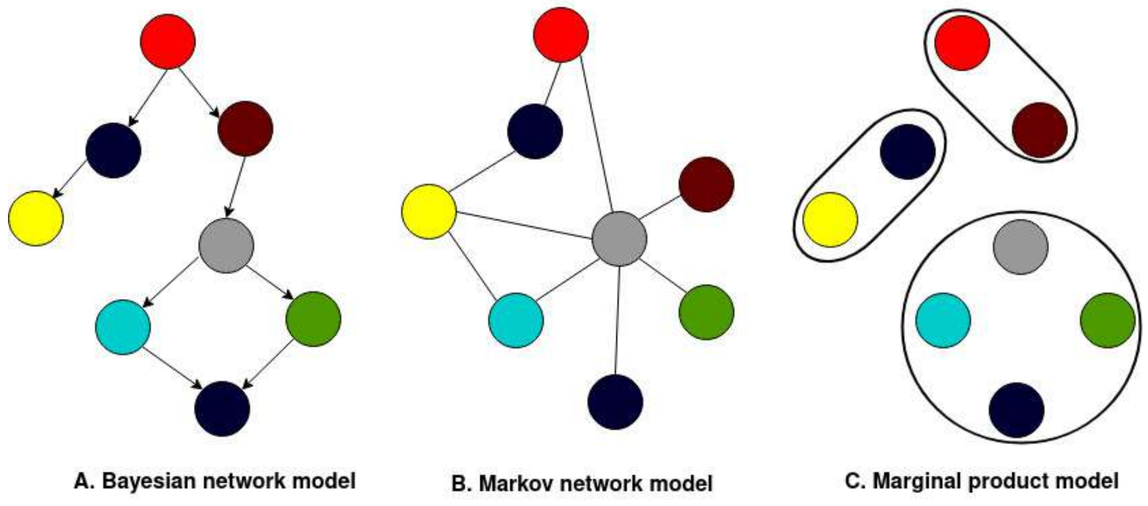

Figure 2.7: Diagram of probability models that represents multiple dependencies

Figure 2.7.C represents another model used in Extended Compact Genetic Algorithm (ECGA) [30] and The dependency-structure matrix genetic algorithm (DMSGA) [31] called Marginal Product

Model. The Marginal Product Model divides variables into clusters, treating each cluster as one variable. Initially assuming all variables are independent, it merges two clusters in each iteration to improve the model [32, 15].

2.5 Estimation of distribution algorithms for permutation problems

In a wide range of real-world problems, permutations offer a versatile means of representing candidate solutions. For instance, in the Traveling Salesman Problem, cities are ordered to find the shortest route. Similarly, job scheduling involves permuting tasks to optimize schedules. Facility location problems explore permutations of potential locations for optimal resource placement. Permutations are also used in network routing [33], game strategy, and various other domains, providing a flexible approach to problem-solving and optimization.

The challenges of employing Estimation of Distribution Algorithms (EDAs) in permutation problems primarily center around the representation, model building, and sampling aspects of the algorithm. In permutation problems, the representation of solutions as permutations requires specialized probabilistic models that respect the combinatorial nature of the problem space. Designing effective models that capture the dependencies and interactions between elements in permutations can be complex. Furthermore, developing efficient sampling methods to generate new candidate solutions based on these models is crucial. The challenge lies in balancing exploration to discover diverse permutations and exploitation to converge toward optimal solutions. Additionally, as the dimensionality of permutation problems increases, the computational complexity of estimating and updating probability distributions escalates, making it imperative to find computationally efficient solutions.

The literature contains numerous EDAs designed to tackle permutation-based problems, with many of these solutions being adaptations of traditional EDAs originally developed for continuous or discrete domains. Nevertheless, some EDAs have been explicitly designed to address permutation problems [34].

2.5.1 EDAs for continuous domains

The first category comprises a group of EDAs that originated from algorithms initially suggested for solving problems in continuous domains. Bosman et al. introduced the continuous Iterated Density Estimation Evolutionary Algorithm (IDEAs) [35] as an approach for addressing permutation problems. Additionally, adaptations of these algorithms for permutation problems include the Univariate Marginal Distribution Algorithm UMDAc [36], Mutual Information Maximization for Input Clustering (MIMICc), and the Estimation of Gaussian Networks Algorithm (EGNAc) [36], which were modified from their continuous domain.

All of the methods within this category are grounded in the Random Keys algorithm [35]. The Random Keys algorithm is a strategy used in optimization and combinatorial problems, introduced in [37]. It involves encoding solutions as real-valued vectors and then transforming those vectors into permutations.

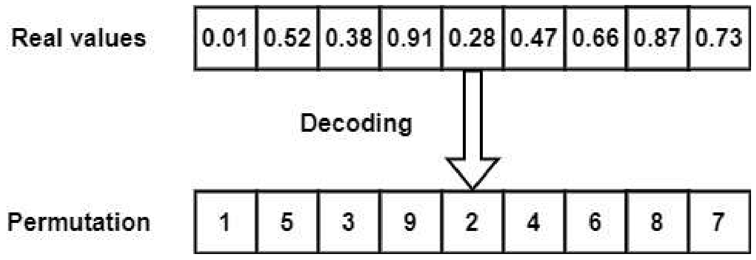

Figure 2.8: Example of random Key representation

Figure 2.8 provides an example of the random key algorithm. In the Random Keys algorithm, the transformation from a real-valued vector to a permutation involves sorting the values within the vector in ascending order. This sorting process assigns a new order to the values, effectively determining the positions of the elements in the resulting permutation. In simpler terms, the position of each element in the permutation is determined by the relative magnitude of its corresponding value in the sorted vector. This step is crucial in converting a continuous real-valued representation into a discrete permutation, making it a fundamental part of the algorithm's operation

While the random key method is capable of producing a feasible permutation from any real-valued vector through a ranking method, it has a significant drawback in terms of redundancy. This technique can result in different vectors of real values producing the same permutation when ranked. [38, 34]. For example two different real-valued vectors $(0.08, 0.74, 0.32, 0.54)$ and $(0.27, 0.85, 0.41, 0.62)$ leads to the same permutation $\sigma = (1, 4, 3, 2)$. Another inefficiency of this strategy arises from the requirement to repeatedly apply an ordering algorithm to transform real-valued representations into corresponding permutations whenever an individual undergoes evaluation [39].

Despite these disadvantages, random key algorithms can still be useful in specific scenarios, and researchers continue to explore ways to mitigate these limitations or develop alternative representations and techniques for EDAs tailored to combinatorial optimization problems.

2.5.2 EDAs for discrete domains

The second category of EDAs developed for permutation-based problems consists of methods initially designed for tackling problems represented as vectors of discrete values. To ensure the generation of valid permutations, the sampling phase is typically modified, with the Probabilistic Logic Sampling algorithm often employed for this purpose [40]. The adapted sampling approach involves setting the probability of previously sampled values to 0 and normalizing the probabilities of the remaining values to a cumulative total of 1.

Within this sampling approach, variables are assigned values in a sequential manner, adhering to a predetermined ancestral order. To sample the i^{th} variable, one must have previously instantiated the $(i-1)^{th}$ variables. However, to transform this process into one that yields a permutation, specific adjustments are necessary. Achieving a permutation entails ensuring that the $i-th$ variable takes on values distinct from those instantiated by the preceding variables. This is accomplished by setting the probabilities associated with previously sampled values to 0 when the $i-th$ variable is being sampled. Simultaneously, the probabilities related to the remaining values are recalibrated and normalized to sum to 1.

Even though this method produces permutations, it's crucial to note that if we modify the probabilities to make it easier to sample permutation individuals, the sampled solutions exhibit a distortion of the information that was kept within the probabilistic model.

Based on this sampling strategy, many proposed algorithms are used to approach permutation-based problems. The dependency-tree EDA (dtEDA)[41] proposed to solve QAP and have good results. Bengoetxea et al. [42] adopted the sampling phase of EBNA [26], the Information

Maximization for Input Clustering (MIMIC), and Univariate Marginal Distribution Algorithm (UMDA) in [43] to ensure the generation of permutations.

2.5.3 EDAs tailored for permutation problems

As discussed above, in numerous real-world scenarios, solutions to critical problems involve arranging elements in a particular order, often represented as permutations. These problems typically encompass some key types of features or constraints that EDAs must effectively capture. The first type concerns the fixed position of a symbol within a permutation, while the second type pertains to the specific order of symbols relative to one another. In certain problems like the traveling salesman problem, the emphasis primarily lies on preserving the relative order of elements. Conversely, in cases such as the quadratic assignment problem, both the relative order and the precise positions of elements hold significance.

Another significant challenge that Evolutionary Distribution Algorithms (EDAs) encounter when addressing permutation problems is the exclusivity mutual constraint imposed by the inherent nature of permutation representation. In these problems, the arrangement of elements in a specific order is not only important but also comes with the constraint that each element can only occupy one position, leading to the need for solutions that satisfy both the desired ordering and the exclusivity of positions. This dual requirement adds a layer of complexity to the optimization process, making it essential for EDAs to effectively handle both aspects in order to find optimal solutions.

Many algorithms have been developed in recent years specifically to solve permutation-based problems. IDEA Induced Chromosome Elements Exchanger (ICE) [**?**], is an algorithm that represents a modification of the IDEA method previously presented by the same authors in [35]. The IDEA can be used to solve permutation problems using the random key method. As discussed above, this strategy has some limitations such the redundancy. To address this challenge, the ICE algorithm was introduced as a solution. Instead of relying on probabilistic sampling to generate new solutions, ICE employs a tailored crossover operator that considers the variable partition learned from the probabilistic model. When constructing a new individual from two parent solutions, ICE randomly selects variable values from a parent within each block. It's important to note that in ICE, the probabilistic model isn't directly utilized for generating new individuals; rather, only the information related to the variable partition is utilized.

Tsutsui et al [44]. introduced the Edge Histogram-Based Sampling Algorithm (EHBSA). This innovative algorithm is centered around the concept of edges, which are connections between two nodes. The model created by EHBSA focuses on learning the interdependencies among these edges and is represented using an Edge Histogram Matrix (EHM).In other words, The algorithm constructs a probabilistic model (in this case EHM) aimed at capturing the relationships between index adjacencies within the chosen individuals at every generation. The EHM is a matrix of size $n \times n$, where n is the problem size. Each element e_{ij} in EHM represents the number of edges between the node i and j. Equation 2.10 calculates the value of bias ϵ added to each element in EHM to control the generation of new solutions, where N is the number of selected individuals and B_{ratio} is a constant defined by the authors.

$$\epsilon = \frac{2N}{n-1} B_{ratio} \tag{2.10}$$

.

To create a new solution, the authors of EHBSA provide two distinct sampling techniques: EHBSA/WT (with template) and EHBSA/WO (without template). The EHBSA/WO sampling method entails selecting the edge histogram matrix in an ordered manner, commencing from position 0 with a randomly chosen node. Next, a roulette wheel is constructed for each row to facilitate the sampling of a new node, akin to the Probabilistic Logic Sampling technique. The process is outlined in Algorithm 2, which summarizes the EHM sampling without a template.

Algorithm 2: The pseudo code of EHBSA/WO [44]

```
1: Set the position counter $p \leftarrow 0$.
2: Obtain the first node $c[0]$ randomly from $[0, L-1]$.
3: Construct a roulette wheel vector $rw[]$ from $EHM^t$ as $rw[j] \leftarrow e^t_{c[p],j}$ for
   $j = 0, 1, \ldots, L-1$.
4: Set to 0 previously sampled nodes in $rw[]$ ($rw[c[i]] \leftarrow 0$ for $i = 0, 1, \ldots, p$).
5: Sample the next node $c[p+1]$ with probability $rw[x]/\sum_{j=0}^{L-1} rw[j]$ using roulette wheel
   $rw[]$.
6: Update the position counter $p \leftarrow p + 1$.
7: If $p < L - 1$, go to Step 3.
8: Obtain a new individual string $c[]$.
```

The sampling with a template method utilizes a randomly selected parent individual from the preceding generation as a template. This template is then divided into more than two segments ($n > 2$), with one segment chosen randomly for sampling, following the same strategy as the previous method. The remaining $(n-1)$ parts are copied into the offspring without any changes.

To validate the effectiveness of EHBSA the authors tested it on TSP. The obtained results demonstrate that EHBSA/WT is more accurate than EHBSA/WO. In [45] is used to solve the Vehicle Routing Problems (VRP). The authors expand upon EHBSA to address the PFSP by introducing an asymmetrical edge histogram model [46, 47].

The same group of researchers introduced another histogram-based algorithm in their work [48], known as the Node Histogram-Based Sampling Algorithm (NHBSA). It employs a probabilistic model represented by a node histogram matrix (NHM). This NHM characterizes the distribution of nodes across the absolute positions within the selected individuals. To put it explicitly, each element e_{ij} in NHM signifies the overall count of element j located at the (absolute) position i within the selected individuals. In other words, it provides a representation of how nodes are spread across different positions within a population. The value ϵ to add to NHM can be calculated using the equation 2.11.

$$\epsilon = \frac{N}{n} B_{ratio} \tag{2.11}$$

As a result, NHBSA offers both NHBSA/WO (without template) and NHBSA/WT (with template) methods, mirroring the options available in EHBSA. In [49], the authors provide a comparative study of various alternative sampling methods for NHBSA, including the substitution of the algorithm's random sampling sequence with a sequential sampling sequence similar to EHBSA. In the original paper, the authors apply the proposed NHBSA and EHBSA to solve three permutation problems: TSP, PFSP, and QAP. The obtained results demonstrate that the algorithm performance varies depending on the problem type. EHBSA excels in PFSP when relative node sequencing matters, while NHBSA outperforms EHBSA when the absolute node position within a string is crucial such as QAP. It is important to note that in all scenarios, the sampling with the template version outperforms the sampling without a template.

Another category of EDAs is specifically tailored for solving permutation problems. These algorithms focus on probability models that rank permutations, such as distance-based exponential models like the Mallows Model and Plackett-Luce's Model [50].

The Plackett-Luce probability model is a probabilistic model for ranking based on the work of Plackett [51] and Luce [52]. Ceberio et al [53] proposed to introduce this ranking model in EDA. The Plackett-Luce model is a ranking model that operates in stages. It breaks down the task of creating a permutation of n items into n consecutive steps. Using a vector of scores denoted as 'w,' in each of these stages, one item is chosen and placed in position i. This selection is made probabilistically, taking into account the scores of the items that have not yet been assigned. The proposed algorithm, Plackett-Luce EDA (PLEDA), was rigorously tested on two challenging permutation problems, the Linear Ordering Problem (LOP) and the Permutation Flow Shop Scheduling Problem (PFSP). The comprehensive analysis of the obtained results revealed that PLEDA exhibits remarkable effectiveness when applied to LOP. However, in the case of PFSP, the algorithm did not perform competitively.

The Mallows model is a distance-based probabilistic model for ranking [54]. The Mallows Model (MM) and Generalized Mallows Model (GMM) represent unimodal distributions that assign probabilities to every permutation within the search space, and these probabilities are contingent upon a distance function applied to permutations. The Kendall-τ distance is the prevalent metric, and it quantifies the dissimilarity between two permutations, $\sigma 1$ and $\sigma 2$, by tallying the total pairwise disagreements between them. In simpler terms, it measures the minimum number of adjacent swaps required to transform $\sigma 1$ into $\sigma 2$.

Ceberio et al. propose the first EDA using the Mallows model (MM) in [55] under Kendall-τ distance for permutation problems. Encouraged by the results of MM-EDA, especially on PFSP, the authors took a further step and proposed the generalized Mallows model (GM) within the context of EDA and abbreviated as GM-EDA [56]. GM-EDA was applied to solve the PFSP with respect to the Total Flow Time (TFT) criterion, and it achieved noteworthy results. Particularly, the hybrid version, HGM-EDA, combined with Variable Neighborhood Search (VNS), showed exceptional performance.

In fact, other distance metrics between permutation can be used in EDAs-based Mallows Models such as Hamming [57], Ulam [58] and Cayley [59]. In [57, 60] an MM-EDAs use the hamming distance proposed to tackle the QAP. Cayley and Kendall'sτ are used in KMM-EDA [61], where kernels of Mallows models are proposed inside EDA. KMM-EDA outperforms MM-EDA and GM-EDA in most 90 used benchmarks of QAP and PFSP. Furthermore, KMM-

EDA when evaluated under the Cayley distance metric, achieves the most favorable outcomes for both problems in regard to average fitness and computational time. In the research proposing the use of MM and GM in the context of EDAs, it has been shown that the choice of distance metric significantly affects the performance of the associated EDA [60, 62]. Ceberio et al. published a review article that summarizes the relation between the three commonly used distances Kendall's-τ, Ulam, and Cayley, and the performance of the EDAs [62].

A Random Key EDA (RK-EDA) suggested in [38] constructs a univariate distribution model similar to UMDAc but with a user-defined variance. To address the redundancy issue associated with other random key-based EDAs, the generated RK vectors are rescaled and normalized within the range of $[0, 1]$. In RK-EDA, a cooling rate parameter is introduced to control the balance between exploitation and exploration. This algorithm has been tested on various permutation problems and has demonstrated promising results in the Permutation Flow-Shop Scheduling Problem (PFSP), both in terms of minimizing makespan [38] and total flow time [63].

2.6 Conclusion

In this chapter, we delved into the captivating world of permutation problems, a class of optimization challenges that encompasses a rich diversity of real-world applications. We explored four prominent permutation problems: the Traveling Salesman Problem (TSP), the Permutation Flow Shop Scheduling Problem (PFSP), the Quadratic Assignment Problem (QAP), and the Linear Ordering Problem (LOP). Each of these problems posed its unique set of complexities and characteristics, pushing the boundaries of computational optimization.

Then, we navigated through the realm of Estimation of Distribution Algorithms (EDAs), particularly those tailored for permutation problems. We witnessed how these intelligent algorithms leverage probabilistic models to capture the underlying structure of problem instances, guiding the search for optimal or near-optimal solutions. Their remarkable ability to adapt and evolve probability distributions allows them to excel in tackling complex permutation problems, often outperforming traditional optimization methods.

Chapter 3

Generative deep learning models

3.1 Introduction

In the past few decades, there has been a significant surge in artificial intelligence knowledge, driven by the growing capabilities of computational systems and the availability of vast datasets across various industries. Machine learning, a prominent branch of artificial intelligence, focuses on refining algorithms to enhance computer and system learning capabilities. Traditionally, these algorithms required manually provided features, a time-consuming and sometimes incomplete process. However, representation learning, also known as feature learning, empowers systems to automatically discover the necessary representations for tasks like feature detection and classification. In essence, representation learning translates input data into meaningful outputs. Deep learning, a subset of representation learning, employs algorithms to model highly abstract concepts within datasets through a layered graph of linear and nonlinear transformations. See 3.1 for an illustrative depiction of these concepts in the hierarchical structure.

3.2 Deep Neural Networks

Deep learning, with its ability to uncover intricate patterns and hierarchies within data, has played a pivotal role in numerous artificial intelligence breakthroughs. Its core principle revolves around the construction of neural networks with multiple layers, allowing it to learn and represent data at various levels of abstraction automatically. These deep neural networks, characterized

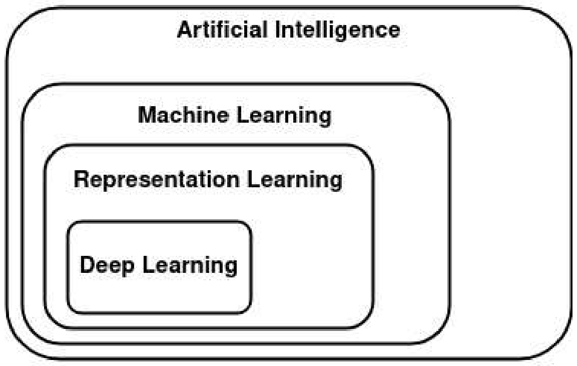

Figure 3.1: A diagram illustrating the hierarchical relationship between deep learning as a subset of representation learning, which in turn is a subset of machine learning, used across various AI approaches.

by their complex architectures and interconnected layers, have demonstrated exceptional performance in tasks like image recognition, natural language processing, and even autonomous decision-making.

The perceptron serves as a foundational unit in neural networks, characterized by a straightforward yet crucial mathematical model. In its essence, a perceptron computes a weighted sum of its input values and applies an activation function to produce an output. Mathematically, this can be represented as follows:

$$\text{Output} = \text{Activation}\left(\sum_i \text{Weight}_i \cdot \text{Input}_i + \text{Bias}\right) \tag{3.1}$$

Each $Input_i$ represents an input feature, and $Weight_i$ signifies the weight associated with that input. The bias term (Bias) allows the perceptron to account for any inherent offset or bias in the data. The activation function introduces non-linearity, enabling the perceptron to capture complex relationships within the data.

Consider a fully connected neural network, as illustrated in Figure 3.2. This network comprises multiple layers, including an input layer, three hidden layers, and an output layer. Each layer consists of interconnected perceptrons, and they collectively form the neural network architecture. Input data, typically represented as a vector, is fed into the input layer, where each perceptron processes the input using the perceptron formula 3.1 mentioned earlier. The output from one layer becomes the input to the next layer, allowing the network to learn and model intricate patterns and representations in the data. Deep neural networks, such as this example, have shown remarkable prowess in various machine learning tasks, thanks to their ability to leverage the foundational concept of the perceptron for extracting meaningful insights from complex datasets.

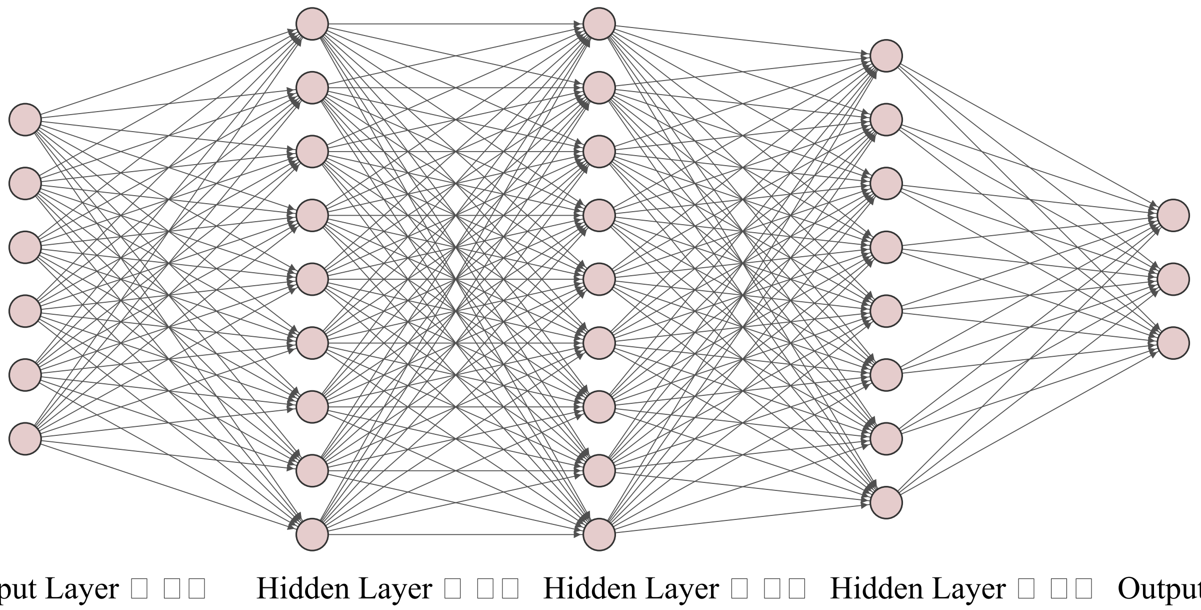

Figure 3.2: An example of deep neural network.

This chapter explores two fundamental pillars of deep learning: discriminative and generative models. These two paradigms represent distinct approaches within the realm of deep neural networks. Discriminative models, such as convolutional neural networks (CNNs) [64] and recurrent neural networks (RNNs) [65], excel in classification and regression tasks, focusing on learning decision boundaries to make accurate predictions. On the other hand, generative models, including Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), delve into the art of data creation, generating new samples that mimic real-world data distributions. Our examination of these models encompasses their architectures, training strategies, applications, and the exciting prospects they offer in shaping the future of artificial intelligence.

3.3 Discrimintive Models

In this section, we delve into the discriminative deep learning models, a pivotal component of contemporary artificial intelligence research. Discriminative deep learning models are engineered to excel in classification and regression tasks, boasting a capacity to learn intricate patterns and relationships from data. Leveraging neural network architectures with multiple layers, these models play a central role in various domains, from computer vision and natural language processing to biomedical research. Through supervised learning techniques and rigorous optimization, they uncover discriminative features, enabling precise classification and regression. This section provides an in-depth exploration of discriminative deep learning methodologies, their applications, and insights into their evolving role in the ever-expanding landscape of artificial intelligence.

The training of discriminative deep learning models involves optimizing their parameters to learn the decision boundary or regression function that maps input data to corresponding target labels. This optimization process is rooted in mathematical concepts and techniques. For classification tasks, the model seeks to estimate the conditional probability distribution $P(Y|X)$, where Y represents the class label, and X denotes the input features.

The choice of loss function depends on the specific task and can include cross-entropy loss for classification or mean squared error (MSE) for regression. During training, input data is fed forward through the network, and the model's predictions are compared to the ground truth labels. Backpropagation is then used to compute gradients of the loss with respect to the model parameters, enabling iterative updates through optimization algorithms like stochastic gradient descent (SGD) or variants such as Adam [66] or RMSprop [67]. This mathematical optimization process continues until convergence, resulting in a trained discriminative model that can accurately make predictions on new, unseen data. The interplay between the mathematical framework and supervised learning techniques underpins the training of discriminative deep learning models, making them powerful tools for various classification and regression tasks.

3.4 Generative Models

One fascinating area within deep learning is Generative Deep Learning, which extends the capabilities of deep neural networks to create entirely new data samples that resemble real-world examples. This field includes notable innovations such as Generative Adversarial Networks (GANs) [68] and Variational Autoencoders (VAEs) [69].

Generative models represent a rapidly evolving field within computer vision research. These models serve as the foundation for unsupervised learning, addressing scenarios where training data, denoted as $\sim pdata(x)$, originates from an obscure data-generating distribution. The core essence of generative models lies in their ability to produce new data samples, $\sim p_{\text{model}}(x)$, that mimic the distribution of the original data. The ultimate objective of any generative model is to generate data samples ($p_{\text{model}}(x)$) that closely resemble the training data ($p_{\text{data}}(x)$). This alignment is best elucidated through a training objective, which seeks to minimize the disparity between the generated data ($p_{\text{model}}(x)$) and the genuine training data ($p_{\text{data}}(x)$). This alignment between $p_{\text{model}}(x)$ and $p_{\text{data}}(x)$ is the fundamental goal that underlies the training process of generative models.

3.4.1 Variational Auto-encoders

Autoencoders (AEs) represent a widely embraced category within the realm of generative models, renowned for their capacity to reduce the dimensionality of high-dimensional data through the use of compact neural networks, all while preserving a significant portion of the original information. At the heart of any autoencoder lies a dual-network architecture, comprising **encoder** and **decoder** networks. The encoder, a series of interconnected layers, plays a pivotal role in taking the input data and compressing it into a reduced-dimensional representation, often referred to as a ”bottleneck”. On the other hand, the decoder assumes the task of reconstructing the input data from this bottleneck representation. Figure 3.3 presents a simple example of an autoencoder, where both the encoder and the decoder are fully connected feedforward networks.

The encoder, represented by the function F_{enc}, maps input data X (comprising n features) to a lower-dimensional representation known as the latent space (Z). This latent space often has fewer dimensions than the original data, capturing essential data patterns and features. The

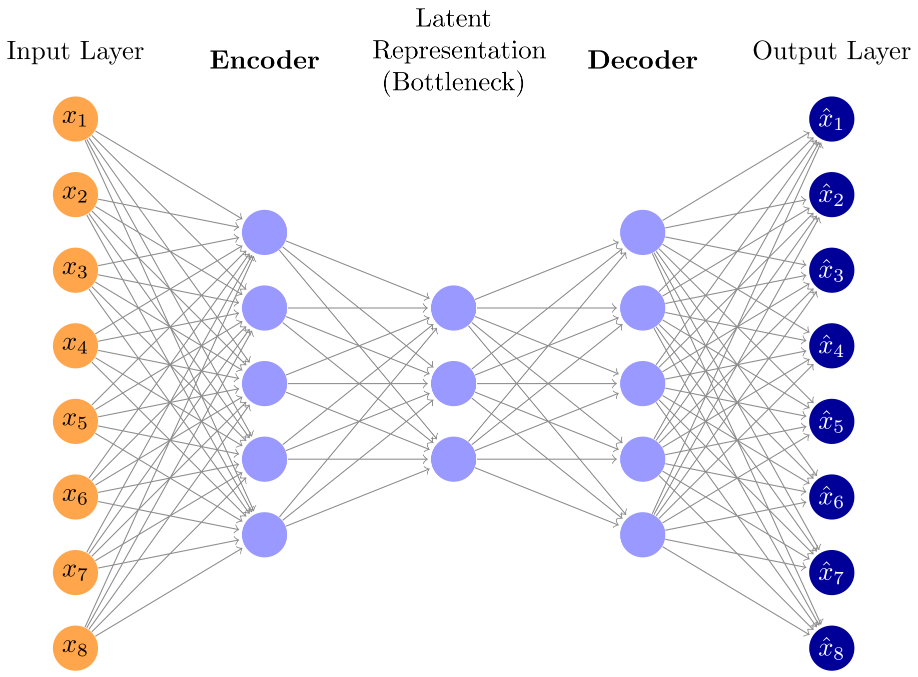

Figure 3.3: Illustrative example of an Autoencoder

encoder generally consists of one or more neural network layers, which can be fully connected or adopt various layer types, such as convolutional or recurrent layers. Conversely, the decoder ($\mathrm{F}_{\mathrm{dec}}$) takes the latent representation Z and reconstructs it back into the original data space, generating X'. Autoencoders are trained by minimizing a chosen loss function that measures the dissimilarity between the input data (X) and the reconstructed data (X'). The primary objective is to achieve accurate reconstruction while learning meaningful data representations within the latent space. Training typically employs backpropagation and gradient descent-based optimization algorithms. This autoencoder framework offers versatility and finds application in diverse areas, including dimensionality reduction, data denoising, and feature learning.

In the following discussion, we delve into one remarkable subclass of autoencoders known as Variational Autoencoders (VAEs), which introduce a probabilistic twist to the traditional autoencoder framework, imbuing it with the capacity for efficient generative modeling and data synthesis.

As shown in Figure 3.4, VAE shares the same architecture as the traditional autoencoder. Variational Autoencoders (VAEs) enhance the capabilities of a conventional Autoencoder (AE) by introducing a Bayesian component that effectively learns the parameters associated with the probability distribution of the data. The primary distinction between an AE and a VAE lies in their core objectives: an AE focuses on learning a compact representation of the input data and its subsequent reconstruction to match the given input. In contrast, a VAE operates as a Bayesian model, where it learns not only the compressed representation derived from the AE but also constructs the parameters that characterize the underlying data distribution. This unique characteristic enables a VAE to sample from this distribution and generate entirely new input data samples. As a result, the VAE assumes the role of a generative model, capable of creating novel data samples. Conversely, an AE primarily excels in data reconstruction without possessing a straightforward generative interpretation [70]. Indeed, the VAE takes a distinct approach by endeavoring to learn the distributions of latent variables through the utilization of mean values and their associated variances. This is in sharp contrast to the traditional autoencoders, which primarily focus on learning deterministic mappings.

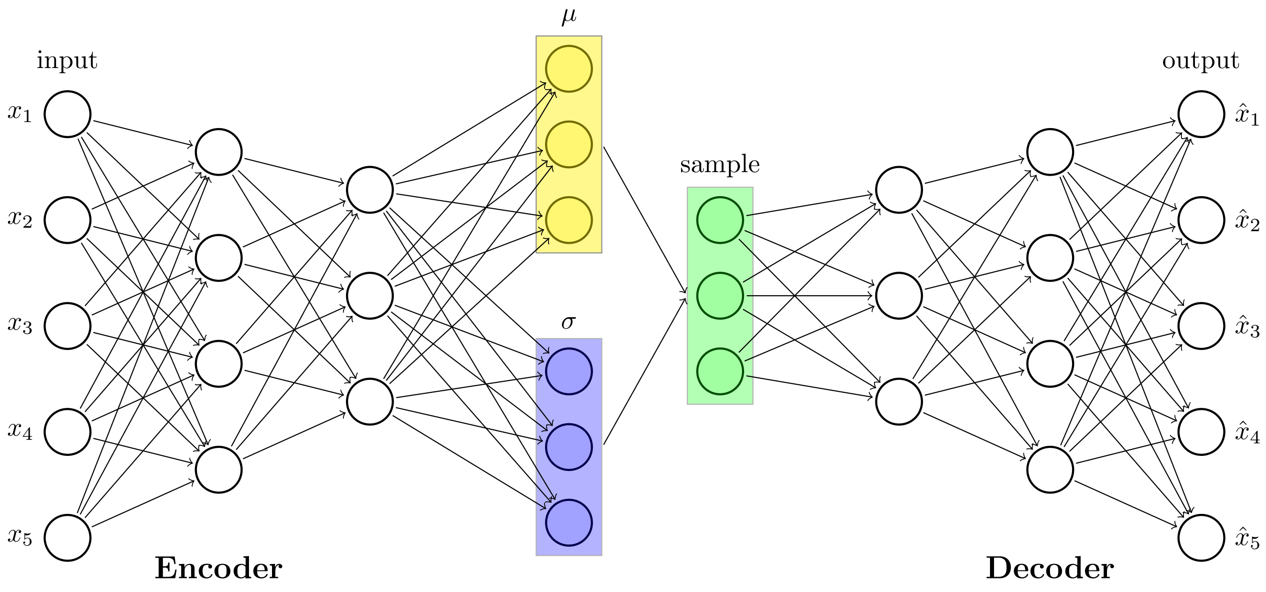

Figure 3.4: Illustrative example of a Variational Autoencoder

In summary, the encoder processes each input and encodes it into two vectors that collectively define a multivariate normal distribution within the latent space:

- μ - representing the mean point of the distribution.
- σ - indicating the logarithm of the variance for each dimension.

The Reparameterization Trick, essential in Variational Autoencoders (VAEs), simplifies the handling of stochastic latent variables. In VAEs, the encoder produces parameters μ and σ that describe a distribution. Instead of directly sampling from this distribution, a fixed Gaussian ϵ is sampled. Then, μ and σ are used to transform ϵ into z through $z = \mu + \sigma\epsilon$. This process ensures that z adheres to the desired distribution. The advantage lies in the determinism of the transformation, allowing gradients to propagate during training. Subsequently, the decoder employs z to generate data samples, rendering VAEs proficient in learning meaningful representations and generating novel data based on the learned distributions [71].

3.4.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) have emerged as a pivotal breakthrough in the field of generative deep learning models and artificial intelligence and introduced by Goodfellow et al. in 2014 [72]. GANs have revolutionized the way we approach generative tasks, including image generation, style transfer, and data augmentation.

GANs, in particular, have gained significant attention for their ability to generate highly realistic and diverse data, ranging from lifelike images to text and audio. The core idea behind GANs involves the interplay between two neural networks: a generator that creates data samples and a discriminator that evaluates their authenticity. Through this adversarial training process, GANs have become indispensable in fields like computer vision, where they have enabled the generation of synthetic imagery indistinguishable from real photographs.

As depicted in Figure 3.5, GANs are models that typically combine two deep neural networks together, **discriminator** D and a **generator** G. The two models are trained in an adversarial manner.

The generator plays the role of a creative craftsman. Beginning with a blank canvas of random noise, it undergoes a transformative process through neural network layers, crafting data samples that strive to capture the essence of real data from a given dataset. Over time, this generator refines its artistry to the point where its creations become nearly indistinguishable from authentic data, aspiring to achieve the pinnacle of realism.

On the other side of the GANs dynamic equation is the discriminator, akin to a discerning judge. It meticulously scrutinizes every data sample, whether generated by the craftsman-like generator or originating from the real dataset. The discriminator's mission is to identify even the subtlest deviations or hints of inauthenticity. As training progresses, this astute critic becomes increasingly proficient at distinguishing between genuine and generated data, providing valuable feedback for both its own improvement and that of the generator.

The term "adversarial" pertains to a scenario in which the discriminator and the generator engage in a competitive process to achieve their individual objectives. Together, through the mechanism of adversarial training, the generator and discriminator engage in a continuous cycle of improvement. The generator strives to outperform the discriminator by producing data that is increasingly convincing, while the discriminator sharpens its discerning eye. This perpetual competition drives GANs toward a point of equilibrium where the generated data emerges as an impeccable mimicry of real data—a testament to the collaborative interplay between these two neural networks at the heart of GANs groundbreaking success.

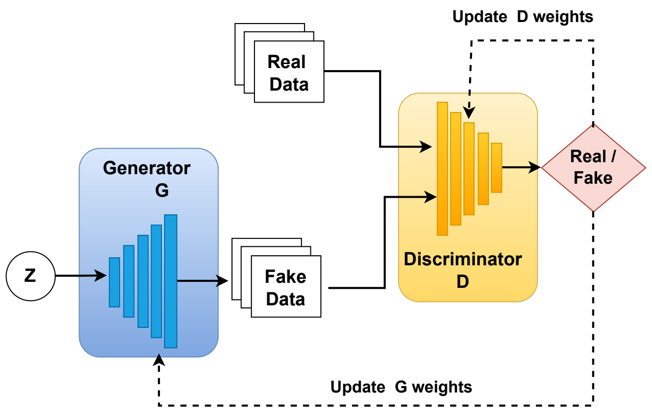

Figure 3.5: GANs architecture

The $minimax$ objective function for a Generative Adversarial Network (GAN) is written as:

$$\min_G \max_D V(D,G) = E_{x \sim p_{\text{data}}(x)}[\log(D(x))] + E_{z \sim p_z(z)}[\log(1 - D(G(x)))] \tag{3.2}$$

The discriminator learns to optimize its objective function in such a way that it aims for a value of 1 (indicating a real instance) for $D(x)$ and a value of 0 (indicating a fake instance) for $D(G(z))$.

The objective of the generator is to minimize its loss function, striving to make the value of $D(G(z))$ as close as possible to 1. The generator aims to minimize the following objective function:

$$V(D, G) = E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{3.3}$$

In this equation, $E_{z \sim p_z(z)}$ represents the expectation over the generator's noise input z. The generator's objective is to minimize this function by producing data $(G(z))$ that is convincing enough to make the discriminator's (D) response $(D(G(z))$ as close to 1 as possible. In other words, the generator strives to create data so realistically that the discriminator is nearly certain it's real.

The discriminator's primary objective is to maximize an objective function, such that it becomes proficient at distinguishing real data $(D(x))$ from fake data $(D(G(z))$. This objective is mathematically represented as:

$$\max_D V(D, G) = E_{x \sim p_{\text{data}}(x)}[\log(D(x))] + E_{z \sim p_z(z)}[\log(1 - D(G(x)))] \tag{3.4}$$

In this equation, $E_{x \sim p_{\text{data}}(x)}$ represents the expectation over real data, $E_{z \sim p_z(z)}$ represents the expectation over the generator's noise input.

The discriminator's objective function formulated in the equation 3.4, denoted as $V(D, G)$, is designed to train the discriminator network (D) to become a discerning judge. It evaluates two types of data: real data (x) sampled from the real dataset (training data) and fake data $(G(z))$ generated by the generator network (G). The goal is to maximize this function.

The first term, $E_{x \sim p_{\text{data}}(x)}[\log(D(x))]$, encourages the discriminator to assign high values (close to 1) to real data in order to correctly identify it as genuine.

The second term, $E_{z \sim p_z(z)}[\log(1 - D(G(x)))]$, motivates the discriminator to assign low values (close to 0) to fake data generated by the generator. This term captures the discriminator's ability to distinguish between real and fake data.

As the generator and discriminator engage in this adversarial game, they iteratively refine their strategies. The ultimate aspiration is to attain a state of Nash equilibrium, where neither player can enhance their situation by altering their strategy. Within the context of GANs, this equilibrium denotes that the generated data closely resembles real data, rendering it exceedingly challenging for the discriminator to distinguish between the two.

In practical terms, GANs persist in this competitive training process until they reach a stable state where the generator produces data that is indistinguishable from real examples. This convergence marks the success of GANs, as they have acquired the capacity to generate data that faithfully mirrors the distribution of real-world data, unleashing a realm of possibilities across a spectrum of applications.

Ever since the advent of GANs, numerous variants have emerged, expanding upon the foundational concept of the original GAN framework. These variations can be broadly categorized into two classes: one that focuses on optimizing architecture, such as CGAN [73], DCGAN [74], and infoGAN [75]. The other concentrates on refining objective functions, such as Unrolled GAN [76], Least-Square GAN [77], EBGAN [78], WGAN [79].

3.5 Conclusion

In this chapter, the focus is shifted toward the exciting domain of generative deep models. Specifically, we explored the Variational Autoencoder (VAE) and Generative Adversarial Networks (GANs) in the context of permutation problems. These generative deep models have emerged as powerful tools for addressing the challenges posed by permutation problems. VAEs have shown promise in capturing the latent structure of permutations, enabling efficient generation and exploration of solution spaces. GANs, with their adversarial training mechanism, have opened doors to creating diverse and realistic permutations, providing fresh perspectives on optimization.

Chapter 4

Introducing Generative Adversarial Networks On Estimation of Distribution Algorithms

4.1 Introduction

Evolutionary algorithms (EAs) are a powerful class of optimization algorithms that mimic the natural process of evolution to find optimal solutions to complex problems. Estimation of distribution algorithms (EDAs) are a subset of EAs that build and sample probabilistic models of promising solutions at each generation. EDAs have been shown to be effective in solving a wide range of optimization problems [18], including permutation-based problems [34, 38], which are problems where the order of the elements in a set matters.

Generative adversarial networks (GANs) are a type of machine learning model that can be used to generate new data samples that match the distribution of a training dataset. GANs have been used in a variety of applications, including computer vision [80], natural language processing [81], and optimization. GANs have been used to estimate the probabilistic model in the field of evolutionary multiobjective optimization [82]. Motivated by this success of GANs, we propose to introduce GANs in EDAs to solve permutation-based problems. The next section introduces the proposed algorithm then we present the results of the experiments conducted. Finally, we end this chapter with a conclusion that contains the main results and some perspectives.

In this chapter, we propose a novel integration of GANs into EDAs to tackle permutation-based problems. In this paradigm, GANs are used to replace the traditional probabilistic model building and sampling phases within EDAs. We also introduce an alternative individual representation technique, which transforms permutations into one-hot matrices.

To evaluate the performance of our proposed algorithm, we use two permutation-based problems: the traveling salesman problem (TSP) and the permutation flowshop scheduling problem (PFSP). The experimental results show that the proposed algorithm outperforms several state-of-the-art EDAs on both problems.

We conclude by discussing the key findings of this work and outlining future research directions.

4.2 Related work

As mentioned in section 2.5.3, specific EDAs are designed for permutation problems. For example, Mallows and Plackett-Luce models have been introduced for permutation-encoded solutions [55, 83, 53]. Notable algorithms, such as the Generalized Mallows-EDA [83], PlackettLuce-EDA [53], Edge Histogram-Based Sampling Algorithm (EHBSA) [84], and Node Histogram-Based Sampling Algorithm (NHBSA) [48], have shown efficacy in solving permutation problems.

On the contrary, a limited number of research papers have explored the integration of Generative Adversarial Networks (GANs) into Estimation of Distribution Algorithms (EDAs). In one such study, detailed in [85], the author proposed the incorporation of GANs within EDA frameworks, although these experiments were conducted on various combinatorial problems that did not involve permutations. Regrettably, the outcomes of these experiments, as reported by the author, fell short of achieving the optimal results.

Another notable endeavor to employ GANs within Evolutionary Algorithms (EAs) is documented in [86], where the authors devised a strategy that combines GANs with a Genetic Algorithm (GA) and applied it to the Permutation Flow-shop Scheduling Problem (PFSP). In this research, the genetic algorithm retained its role as the primary component, with the GAN serving a supplementary purpose. Specifically, the GAN was employed to enhance the genetic algorithm's performance by aiding it in circumventing local optima and diversifying the population through the introduction of generated samples. The author of this study asserted that the hybrid GAN-GA approach yielded superior outcomes compared to traditional genetic algorithms.

4.3 Proposed GAN-EDA Algorithm

Our proposed algorithm, GAN-EDA, simulates the traditional EDA framework. The algorithm begins by randomly generating an initial population. Selected individuals are then normalized between [0, 1] and labeled as real data. GAN-EDA trains the models to mimic the distribution of the selected individuals and subsequently uses the trained generator model to produce new individuals (see Algorithm 3). This process is repeated in each iteration of the algorithm. In essence, the generator creates an implicit probabilistic model of the dataset during the training phase (using selected individuals) and generates a new population during the sampling phase.

Algorithm 3: Pseudocode of the Proposed GAN-EDA Algorithm

```
1 Generate the initial population;
2 while Termination criteria are not met do
3     Select promising individuals and normalize their genes between [0, 1];
4     Train the GAN using the selected individuals;
5     Generate new samples using the trained generator;
6     Apply random keys to obtain individuals;
7     Replace the old population with the new individuals;
8 end
```

We employ the generator to generate a collection of offspring categorized as fake data. Subsequently, the discriminator is trained on both real and fake data, with the objective of distinguishing between them. The generator is then adjusted based on the generated samples, with its input being a random vector drawn from a normal distribution.

In each generation, the selected individuals form the training dataset (real data). To train the discriminator, we normalize the permutation vectors before feeding them to the discriminator. The generator cannot directly produce a permutation due to its output being a vector of real values. To obtain a permutation, we use the traditional Random Keys algorithm, ranking the positions of the generated values. We train the GAN for a specified number of epochs, and

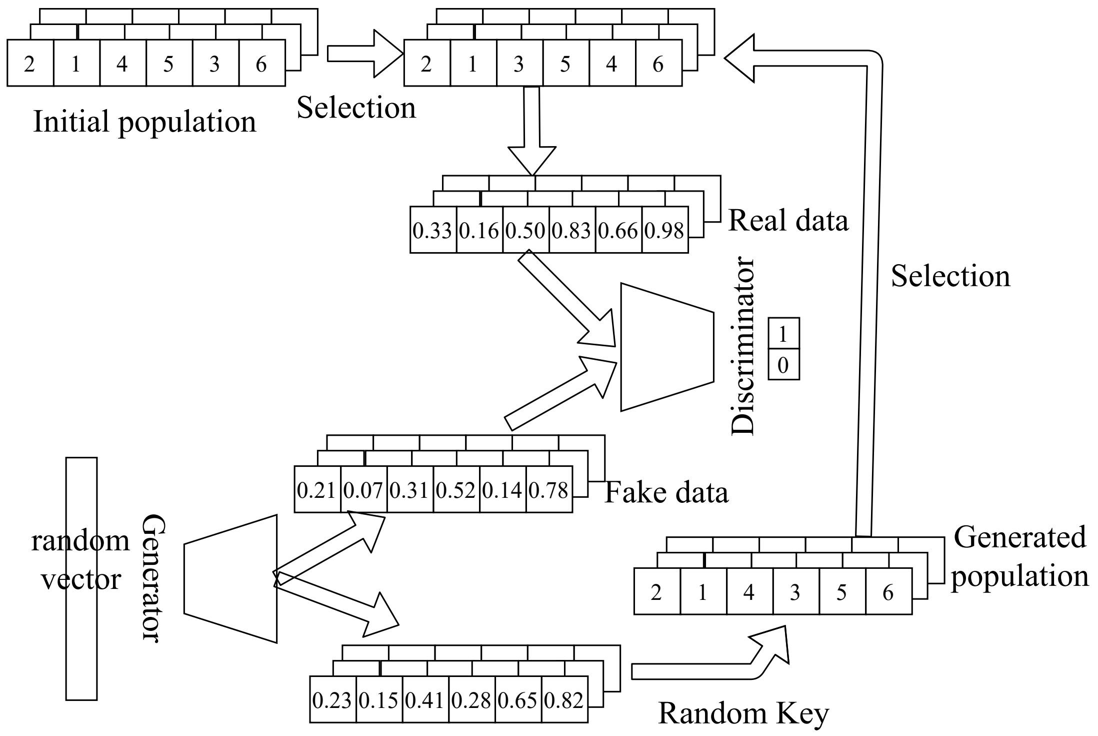

Figure 4.1: General scheme of GAN-EDA

afterward, we use the generator to create the next generation, repeating this process as needed. An overview of the algorithm's workflow is depicted in Figure 4.1.

To improve the performance of GAN-EDA, we introduce a hybrid variant called HGAN-EDA. HGAN-EDA incorporates the 2-opt local search algorithm to refine the solutions generated by GAN-EDA. The 2-opt algorithm is a powerful heuristic that iteratively swaps pairs of edges in a solution tour to reduce the overall tour length or cost [87].

In the context of the 2-opt local search algorithm (See algorithm 4), the primary objective is to explore the neighborhood of the current solution in search of an improved solution. Should such an improved solution be encountered, it promptly replaces the existing solution, and the search process continues. Conversely, if no better solution is found within the local neighborhood, it implies that a local optimum has been reached.

Algorithm 5 presents the swap neighborhood structure used in the 2-opt algorithm.

Algorithm 4: Two-Opt Algorithm

```
Input: tour: A list representing the TSP tour
Output: tour: A new shorter TSP tour
improved ← true;
while improved do
    improved ← false;
    best_distance ← calculateTotalDistance(tour);
    for $i \leftarrow 1$ to $length(tour) - 2$ do
        for $j \leftarrow i + 1$ to $length(tour) - 1$ do
            new_tour ← twoOptSwap(tour, $i$, $j$);
            new_distance ← calculateTotalDistance(new_tour);
            if new_distance < best_distance then
                tour ← new_tour;
                best_distance ← new_distance;
                improved ← true;
            end
        end
    end
end
return tour;
```

Algorithm 5: 2-Opt Swap Procedure

```
Input: route: A list representing the route, $v1, v2$: Indices of vertices to swap
Output: new_route: A new route with vertices swapped
1  $new\_route \leftarrow []$;
2  for $i \leftarrow 0$ to $v1$ do
3      append($new\_route, route[i]$)
4  end
5  for $i \leftarrow v2$ to $v1 + 1$ step $-1$ do
6      append($new\_route, route[i]$)
7  end
8  for $i \leftarrow v2 + 1$ to length($route$) $- 1$ do
9      append($new\_route, route[i]$)    ▷ Take route[v2+1] to route[end] and add them in order
10 end
11 return $new\_route$;
```

In the experimental setup, we adopt the first-improvement principle. This means that upon discovering the first superior solution during the search within the neighborhood, we promptly accept and implement it as the replacement for the current solution. This approach ensures that the local search algorithm efficiently navigates through the solution space, making immediate improvements whenever possible, thereby facilitating the quest for optimal or near-optimal solutions.

4.3.1 Experimental

In this section, we examine the outcomes of the proposed algorithm on specific TSP instances. To achieve this, we have employed a discriminator and a generator, both featuring a Multilayer Perceptron architecture with two hidden layers. Notably, the output layer of the generator and the input layer of the discriminator share the same size, which corresponds to the size of the individuals. The used parameter settings for the GAN are detailed in Table 4.1.

Table 4.1: Parameters for Both the Proposed Algorithm and the GANs Models.

Parameter	**Value**
Maximum number of generations	100
Population size	500
Selection size	50
Number of epochs	50
Discriminator hidden layers	2
Generator hidden layers	2
Learning rate	1e-3
Activation function (output layer of D)	Sigmoid
Activation function (output layer of G)	Tanh
Activation function in hidden layers	Relu
Batch size	32

Table 4.2: Comparison of Results Achieved by the Proposed Algorithm with the Best-Known Solutions.

Benchmark	**Problem size**	**Best known**	**With 2-opt**	**Without 2-opt**
gr17	17	2085	2128.2	2808.8
bayes29	29	2020	2806.4	4389.8
fri26	26	937	1110	1334.6
DANTZIG42	42	699	929.2	1327.4

In Table 4.2, we can observe the best-known solution values for various benchmarks, along with the results achieved by our proposed algorithm with and without the 2-opt local search method. The benchmarks of TSP instances included are gr17, bayes29, fri26, and DANTZIG42. Figure 4.2, provides a visual representation of the experiment results, offering a comprehensive view of how our proposed algorithm performs compared to the best-known solutions across different benchmarks.

It is noteworthy that our proposed algorithm features two variants: one utilizing a 2-opt

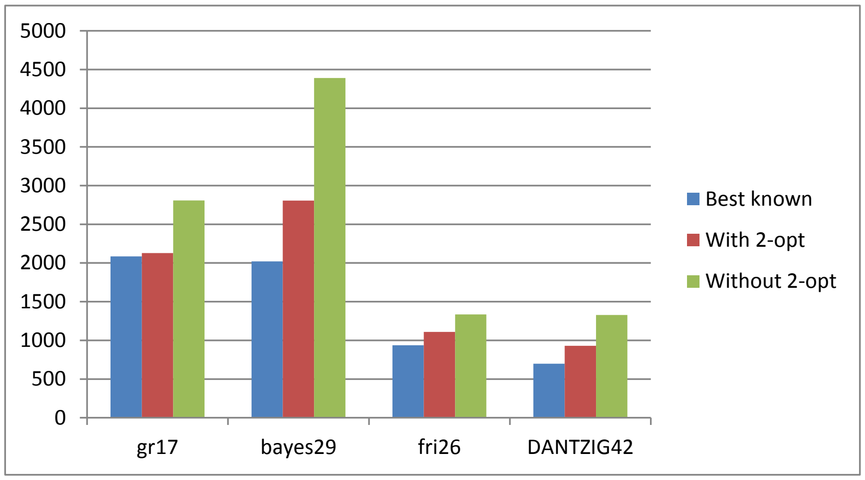

Figure 4.2: Experiments result for the used benchmarks and the best known solutions

local search method and the other without it. Upon analyzing the results, it becomes evident that incorporating the 2-opt algorithm leads to superior solution quality for all the benchmarks studied.

Without the 2-opt local search method, our proposed algorithm still delivers competitive results when compared to the best-known solutions. While the solutions obtained in this variant are not as superior as those with the 2-opt optimization, they demonstrate the algorithm's (in fact the trained generator) capability to produce high-quality. This aspect is particularly significant, as it suggests that the proposed algorithm's initial solutions are already close to optimal, and applying additional local search methods can further refine them. The results obtained without 2-opt showcase the algorithm's inherent strength in generating quality solutions, making it a versatile and robust tool for addressing a wide range of permutation-based problem

These results signify the effectiveness of the 2-opt local search method within our proposed algorithm. It demonstrates how optimizing the solutions obtained from the initial algorithm run can lead to substantial improvements in solution quality. These findings underscore the importance of considering and incorporating the hybridization of EDAs with local search techniques when tackling complex optimization challenges.

4.4 An alternative individual representation for GAN-EDA

4.4.1 Proposal

In this section, we will comprehensively explore our proposed algorithm, covering every step from how individuals are represented to the method of generating offspring.

4.4.2 Representation of individuals

A common way to represent a permutation is as a vector of discrete numbers, with each number denoting a unique position within the permutation. For example, the permutation $\sigma(2, 3, 1, 4)$ can be represented as the vector [2,3,1,4]. However, this representation is not suitable for training GANs, because GANs require real-valued inputs and outputs.

One way to transform a permutation into a real-valued vector is to use a one-hot matrix. A one-hot matrix is a binary matrix of $(n * n)$ where each row represents a possible value and each column represents a position in the permutation. The only non-zero element in each row is the element that corresponds to the value at that position in the permutation.

For example, the permutation $\sigma(2, 3, 1, 4)$ can be represented by the following one-hot matrix:

0	1	0	0
0	0	1	0
1	0	0	0
0	0	0	1

In this matrix, the first row represents the position of the number 2 in the permutation, the second row represents the position of the number 3 in the permutation, and so on.

This representation offers an advantage over the natural permutation representation by retaining all information pertaining to the permutation, including the order of its elements

In our proposed algorithm, we use one-hot matrices to represent the individuals (permutations) in the population. This allows us to train a GAN to generate one-hot matrices, which we can then convert back to permutations using a simple decoding rule.

4.4.3 Proposed algorithm

The main idea behind the suggested method is depicted in Figure 4.3. The initial population and GAN weights are randomly initialized. Next, a group of individuals is selected based on their fitness. These selected individuals are transformed into matrices using the individual representation method described earlier. The transformed individuals into one-hot matrix are labeled as real data and utilized to train the GANs, alongside generated data from the generator, which is considered fake data. Once the generator is adequately trained, it is employed to generate the next generation after a specified number of epochs.

Algorithm 6: The proposed Algorithm

```
1 $P_0 \leftarrow$ create M individuals at random;
2 for $gen \leftarrow 0$ to $maxgeneration$ do
3     Selected set $\leftarrow$ Select $N < M$ best solutions;
4     Real Data $\leftarrow$ Transform Selected set to matrices;
5     Train the GAN with Real Data (Matrices);
6     Offspring $\leftarrow$ Generate $M$ new offspring by updated Generator;
7     $P_{i+1} \leftarrow$ Update $P_i$;
```

After a certain number of epochs, a collection of matrices that mimic the distribution of the training matrices are generated using the trained generator. Subsequently, we apply the reverse operation to obtain new individuals from these generated matrices.

4.4.4 Offspring reproduction strategy

The offspring generation is exclusively handled by the generator. The generator takes a random input z, sampled from a normal distribution, and transforms it into a matrix L with dimensions R^{N*N}. In this matrix, each row represents a vector of probabilities for obtaining a number in each position. For example, if the first row of the generated matrix is $(0.25, 0.73, 0.41, 0.29)$, it corresponds to the one-hot matrix $(0, 1, 0, 0)$. Subsequently, we apply the inverse operation of individual representation to derive offspring from this matrix. Finally, we utilize the same 2-opt algorithm used in HGAN-EDA to further enhance the obtained results.

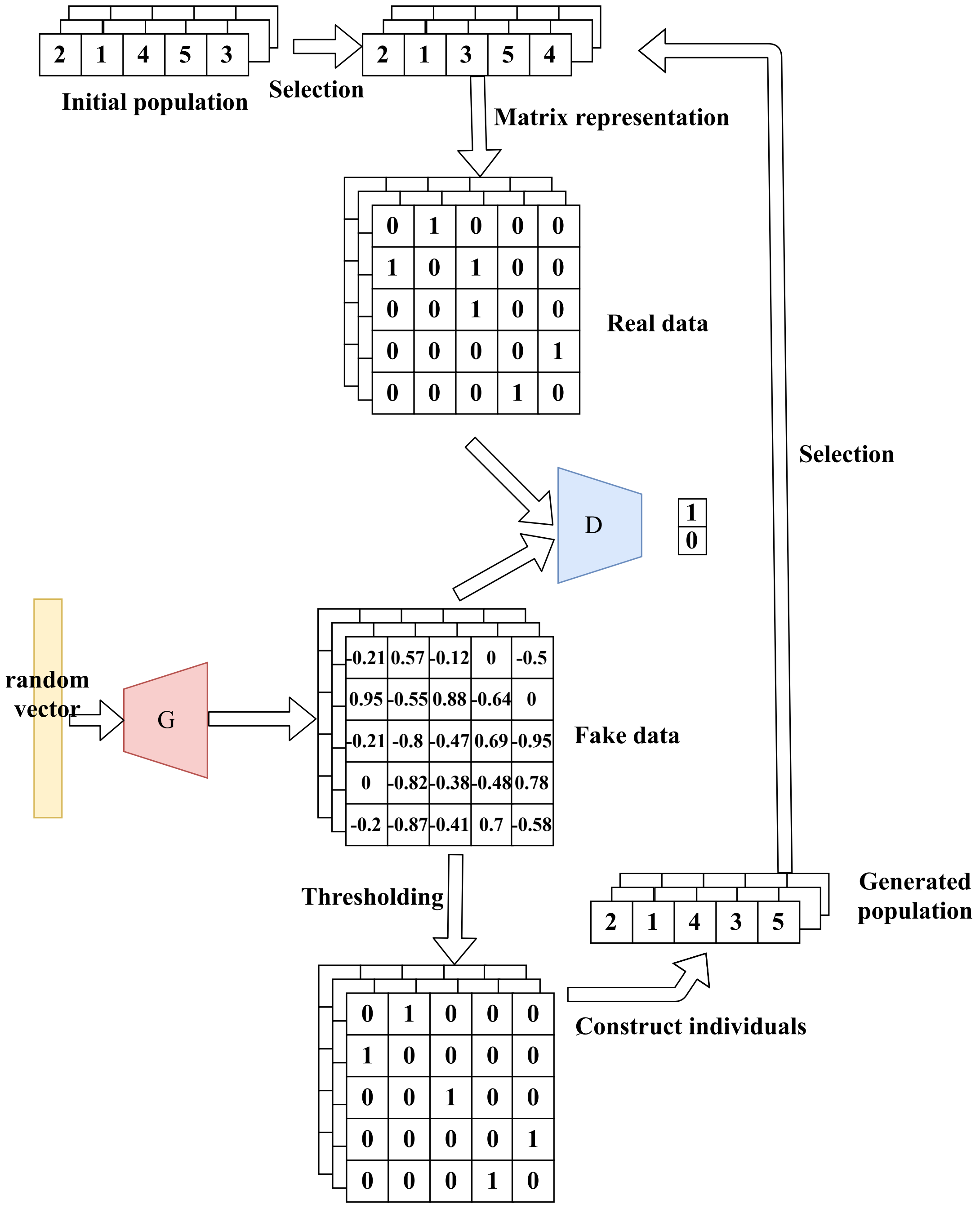

Figure 4.3: General scheme of the GAN-EDA with one-hot matrix representation

4.4.5 Experiments and results

4.4.6 Experiments

To evaluate our proposed algorithm, we conducted a series of experiments using small instances of two widely recognized permutation problems: the Traveling Salesman Problem (TSP) instances from TSPLIB[88] and the Permutation Flow Shop Scheduling Problem (PFSP) instances from Eric Taillard's benchmarks[89].

Problem	Used instances
TSP	fri26, bays29, gr17, gr21, gr24
PFSP	20×5-1, 20×5-2 , 20×10-1, 20×10-2

We have implemented the GAN models with the Pytorch library and integrated the evolutionary operations of EDA) using the DEAP library[90]. Table 5.2 outlines the parameters for the GANs that were determined as the best through experimentation.

Parameter	Value
Epochs	50
Discriminator hidden layers	1
Generator hidden layers	2
Learning rate	1e-3
Generator output activation function	Tanh
Discriminator output activation function	Sigmoid
Hidden layers activation function	Relu
Batch size	32

Table 4.3: Model Architectures and Parameter Settings

4.4.7 Results and discussion

This section showcases the experimental outcomes of the algorithm under consideration. In Table 4.4, we present the minimum, maximum, and average values resulting from ten separate runs of the proposed algorithm for each selected instance. Any values shown in bold signify the most optimal known solution.

The results in Table 4.4 reveal the strong performance of our method. It effectively produces optimal solutions in 3 out of 5 Traveling Salesman Problem (TSP) benchmark instances, approaching optimality in the others. While its performance in the Permutation Flow Shop Scheduling Problem (PFSP) benchmarks is slightly less competitive, it remains acceptable.

Problem	Selected instance	Best known	Min	Max	Mean
TSP	fri26	937	**937**	1019	977
	bays29	2020	2034	2104	2065
	gr17	2085	2090	2142	2101.1
	gr21	2707	**2707**	2801	2755
	gr24	1272	**1272**	1346	1305.6
PFSP	tai20-5-0	1278	1297	1307	1297.9
	tai20-5-1	1359	1367	1395	1380.9
	tai20-10-0	1582	1654	1709	1683.1
	tai20-10-1	1659	1740	1781	1762

Table 4.4: The Results of Experiments

Instances	Random key representation		Matrix representation
	Without 2-opt	With 2-opt	
fri26	1334.6	1110	**937**
bays29	4386.8	2806.4	2034
gr17	2808.6	2128.4	2090
gr21	5491.6	3555.5	**2707**
gr24	/	/	**1272**

Table 4.5: Comparing Results between Random Key Representation and Matrix Representation

Table 4.5 compares outecomes of our proposed algorithm, using the one-hot matrix representation, with our previous work with random key coding of permutations, considering both 2-opt and non-2-opt variants. The results indicate that our new algorithm significantly outperforms the prior method in all TSP instances. This suggests the potential of our approach for addressing a broad range of TSP instances, making it a promising solution for real-world optimization challenges.

The comparison of the results shown in Table 4.5 demonstrates that the matrix representation of permutations enhances the information captured by the GANs from the population distribution. This is likely due to the fact that the matrix representation preserves more information about the permutation than the natural representation, such as the order of the elements in the permutation.

Nonetheless, the matrix representation's drawback lies in its extensive computational time, mainly due to the substantial number of parameters that need to be trained in the GANs. This is because the matrix representation requires a larger number of parameters to represent the same amount of information as the natural representation.

Despite this disadvantage, we believe that the benefits of using the matrix representation outweigh the costs. The matrix representation allows us to train GANs to generate better permutations, which can lead to improved performance on optimization problems.

4.5 Conclusion

In this chapter, we proposed a novel estimation of distribution algorithm (EDA) based on generative adversarial networks (GANs) for solving permutation-based problems. The primary challenge of an EDA lies in its approach to modeling the best solutions found in each generation. In our study, we harnessed the capabilities of a GAN for this purpose, as GANs excel in modeling in continuous spaces. However, the utilization of GANs to learn and generate permutations remains a largely uncharted territory.

We introduced a novel methodology for training a GAN using a set of permutations and subsequently leveraging it to generate fresh permutations that guide the optimization process. The experiments conducted in this research aimed to assess the GAN's capability to generate samples that exhibit similar characteristics to the input permutations and effectively utilize them in the optimization process. The results obtained for the benchmark problems are not satisfactory when using a permutation representation, but they show promise when we employ the matrix representation. However, further exploration is warranted, particularly regarding the number of permutations utilized to train the GAN.

One limitation of our proposed GAN-EDA is the significant execution time, particularly for the matrix representation, due to the larger model architecture. However, our focus is on the ability of GANs to capture the distribution over permutation space, rather than on algorithm complexity or execution time.

We intend to evaluate our proposed method on larger instance sets and examine a number of GAN variants. In future research. Furthermore, we are contemplating the implementation of variational autoencoders (VAEs) within EDAs as a probabilistic model estimator.

Overall, our proposed algorithm is a promising new approach to solving permutation-based problems. We believe that further research in this area has the potential to lead to significant advances in the field of optimization.

Chapter 5

An Estimation of Distribution Algorithm for Permutation Flow Shop Scheduling Problem

5.1 Introduction

In recent years, numerous specialized algorithms have emerged to tackle permutation-based problems. Notably, Tsutsui et al. introduced the Edge Histogram-Based Sampling Algorithm (EHBSA) [44]. This innovative algorithm models the interdependence of edges using an edge histogram matrix. It features two key sampling methods: EHBSA/WO (without template) and EHBSA/WT (with template), providing distinct strategies for generating new individuals.

Similarly, the Node Histogram-Based Sampling Algorithm (NHBSA) [48], also proposed by Tsutsui and colleagues, employs a node histogram matrix to capture the distribution of nodes across absolute positions in selected individuals. NHBSA offers versatility by supporting the same two sampling techniques found in EHBSA. These algorithms have already been discussed in detail in the previous chapter.

However, this approach can present challenges because the relationships between elements and their positions play a vital role in the algorithm's performance. Additionally, there is currently no established method for assigning a node to a specific position, often leading to sequential or random assignments. Surprisingly, to our knowledge, no existing EDA for permutation problems has effectively tackled this issue.

Several existing algorithms designed for permutation-based problems primarily concentrate on sampling nodes instead of positions. This method, however, poses significant challenges, as the interactions between positions and elements play a vital role in the algorithm's performance. Furthermore, there is currently no standardized technique for mapping a node to a particular position. Surprisingly, to date, there are no known EDAs designed specifically for permutation problems that adequately address this operation.

The Permutation Flow-Shop Scheduling Problem (PFSP) has attracted considerable interest in the research community due to its wide range of applications in manufacturing, production planning, textiles, and logistics [10, 91].

The objective of PFSP is to schedule a given set of jobs on a set of machines, aiming to minimize a specific performance metric. Among the various performance measures, the makespan is the most commonly employed [11, 92]. Makespan refers to the total time required to complete all jobs across all machines, making it an important parameter in the context of PFSP. The problem has been extensively researched and is commonly used as a benchmark for assessing the effectiveness of optimization algorithms [11, 63].

Minimizing the makespan in PFSP has been addressed through various methodologies. Exact algorithms, such as the branch and bound algorithm [93, 94], have been applied, but their effectiveness diminishes as the problem scales due to the NP-hard nature of PFSP. Consequently, heuristic and metaheuristic methods have emerged as prominent approaches to tackle this computational challenge. Several heuristic algorithms have been proposed to provide approximate solutions for PFSP [95, 96, 97, 98].

Furthermore, the realm of combinatorial optimization has witnessed the advent of metaheuristic techniques, which offer a diverse set of strategies for PFSP. These include genetic algorithms [99, 100], simulated annealing [101], estimation of distribution algorithms [56, 102, 38], and particle swarm optimization [103, 104]. Notably, in [105], a parallel metaheuristic approach was specifically crafted to address the challenges posed by PFSP.

In this chapter, we introduce a novel EDA designed for solving permutation-based problems, with a specific focus on the Permutation Flow-Shop Scheduling Problem (PFSP). Our proposed algorithm, known as PGS-EDA, incorporates a unique sampling technique. This innovative sampling method is a key contribution of this work, allowing for a more profound exploitation of the information captured by the probabilistic model. The sampling process is guided by a sequence vector (SV) created during the modeling phase, ensuring that each element is assigned to a particular position based on its order within the SV. This guided approach empowers the algorithm to efficiently explore the search space by exploiting the valuable information within the model.

Moreover, our proposal employs a dynamic updating mechanism to adapt the distribution model as the search progresses. This dynamic adjustment enables the algorithm to strike a balance between exploration and exploitation, resulting in effective and efficient optimization. Through a comprehensive set of experiments on benchmark instances of the PFSP, we demonstrate that the proposed PGS-EDA consistently discovers high-quality solutions and surpasses the performance of existing state-of-the-art algorithms.

5.2 Problem Definition

Designing an EDA becomes particularly challenging when working with permutation representations due to the mutual exclusivity constraint, which enforces that each element can only appear once in a permutation ($P(\sigma : \sigma(i) = \sigma(j))) = 0 \quad \text{for all} \quad i \neq j$) [106, 34]. This constraint poses a significant hurdle in creating effective algorithms for solving permutation problems, necessitating unique approaches to modeling and sampling [34]. Researchers have responded to this challenge by adapting EDAs, originally designed for continuous or discrete domains, into specialized methods tailored to address permutation problems.

The primary goal of the Permutation Flow-Shop Scheduling Problem (PFSP) is to determine the optimal job permutation that minimizes the makespan or total flow time. In this work we are interested in the makespan criterion, which is defined as the completion time of the last job on the last machine ($C_{n,m}$), and can be mathematically expressed using the following equations:

$$C_{\text{max}} = C_{n,m} \tag{5.1}$$

Or

$$C_{\max} = \max\{C_{n,m-1}, C_{n-1,m}\} + P_{n,m} \tag{5.2}$$

The makespan is calculated by summing the processing time of the last operation of the final job on the last machine (denoted as $P_{n,m}$) with the maximum value among the completion time of the preceding job on the same machine ($c_{\sigma\langle n-1\rangle,m}$) and the completion time of the current job on the machine used in the previous operation ($c_{\sigma\langle n\rangle,m-1}$).

Algorithm 7: Makespan Calculating

```
Input: Permutation $\sigma = [\sigma_1, \dots, \sigma_n]$
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 $C_{\max}$
```

Algorithm 7 describes how to calculate the makespan for a specific solution (permutation). This algorithm initializes a completion time matrix (C) and iterates through the elements, progressively calculating the makespan $C_{\max}$. In essence, it efficiently determines the optimal job arrangement that minimizes the makespan, a crucial objective in PFSP.

5.3 The Proposed PGS-EDA Algorithm

In this section, we provide a comprehensive overview of our proposed algorithm. The discussion commences with an explanation of individual representation and initialization. Subsequently, we introduce our novel approach to modeling and sampling. Finally, we delve into the details of population evolution across generations.

The proposed PGS-EDA algorithm follows the fundamental EDA framework outlined in Section 2.4. The algorithm's high-level steps are summarized in Algorithm 8.

Algorithm 8: PGS-EDA

```
Input:
$Pop_s$: Population size
$Sel_s$: Selection size
Output:
The best solution
1  $P_0 \leftarrow$ Generate $Pop_s$ individuals at random;
2  $t \leftarrow 0$;
3  while Stopping Criteria not satisfied do
4      Select $Sel_s$ fittest individuals from $P_t$;
5      Construct matrix model $M$ using $Sel_s$;
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 $P_t$;
```

5.3.1 Population initialization and solution representation

In evolutionary algorithms, random initialization is a frequently utilized technique for creating the initial population in evolutionary algorithms. This method involves generating the initial population in a random and uniform manner to ensure diversity, as suggested by previous research [107]. In our study, individuals are depicted as job permutations, each representing a unique job arrangement. When scheduling tasks, jobs are assigned to machines in the sequence determined by the permutation.

5.3.2 Modeling

In each generation, individuals are chosen based on their fitness, with each individual represented by a permutation σ_i as defined in Equation (5.3):

$$\sigma_i = (\sigma_i(1), \sigma_i(2), \sigma_i(3), ..., \sigma_i(n)) \tag{5.3}$$

Where n signifies the problem size, the proposed algorithm constructs a probabilistic model that characterizes the frequency with which a specific element occurs at a particular position across the entire population. In simpler terms, it quantifies how often the element i is positioned at location j.

The model takes the form of a matrix denoted as M, consisting of n^2 elements. It is defined by the following equation:

$$m_{ij} = \sum_{i=1}^{k} \alpha_{ij} + \varepsilon \tag{5.4}$$

Here, k represents the size of the selected individuals, and α_{ij} is a function for counting the occurrences of elements, which is defined as:

$$\alpha_{ij} = 1 \quad if \quad \sigma_i = j, \tag{5.5}$$

Additionally, ε serves as a user-defined constant. It plays a pivotal role in the algorithm during the sampling phase by enhancing numerical stability while preventing mathematical complications when dealing with zero-valued elements.

Hence, we can represent our model (M) as a square matrix with dimensions $n \times n$, described as follows:

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

Incorporating the addition of ϵ to all elements of M:

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

The derivation of the sequence vector (SV) from the matrix model (M) is a crucial step in the algorithm, as it guides the subsequent sampling process. The SV is a vector of size n, where n is the number of elements in the matrix model. The SV represents an element permutation, where the elements are ordered according to their frequency of appearance at the same position across all individuals.

To derive the SV, the algorithm first identifies the element with the highest frequency in a given position among the chosen individuals. Subsequently, this element is positioned at the relevant location in the SV. This pattern is repeated for each element in the matrix model, guaranteeing that the most prevalent elements at each position are situated at the start of the SV.

The SV is a powerful tool for guiding the sampling process, as it ensures that the most important elements are more likely to be selected for the offspring. This can lead to a significant improvement in the performance of the algorithm.

Here is an example of how the SV can be extracted from the matrix model. Let $P(k)$ denote the population at 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 model M(k) is updated at each generation to reflect the current state of the population. This is done by considering how often each element appears at each position in the strings within the population. A small value, ϵ (in this example $\epsilon = 0.4$) is added to all elements of the matrix. The matrix model is then used to sample positions for the offspring in the next generation.

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

The sequence vector SV can be extracted from the matrix M. In this example, the SV will be (2,3,1,4,5,0) since the element 2 is repeated three times at position 2, therefore, it is assigned to the first position in the SV. The element 3 is repeated thrice at position 5. so it is placed at the second slot in the SV, and so on. The element 0 appears only once in each position.

5.3.3 Sampling

Unlike other algorithms such as NHBSA, EHBSA, or UMDA, which follow a sequential element selection process. Diverging from the conventional approach that proceeds linearly From the initial position to the final position, our suggested method uses a distinctive path. Instead of focusing on selecting elements for positions, we prioritize the positions themselves.

Central to our algorithm is the Sequence Vector (SV), which dictates the ranking in which elements will be positioned in the new individual. During the sampling process, we draw elements from the SV and decide the most suitable position for each one. Our goal is to enhance the organization of items inside the individual by allocating places depending on their ranking in SV. You can find the detailed sampling algorithm in Algorithm 9. This approach allows us to potentially enhance the quality of the offspring arrangement compared to traditional methods.

Algorithm 9: The proposed algorithm for position sampling

```
Input: M : The matrix model, $SV$: The sequence vector
Output: $Off$ : Offspring
1 $SV$ ←Sawp($SV$, $n/10$); // Apply $n/10$ swap operations to $SV$
2 for $element\ e\ in\ SV$ do
3     $PV$ ← Normalize M[e] ; // The column $e$
4     $p$ ← Sample position from $PV$ ;
5     $Off[p]$ ← $e$ ;
6     $M[p]$ ← 0; // The row $p$
7 end
8 return $Off$;
```

Algorithm 9 proposes a new method for offspring generation (sampling) using both a matrix model M and a sequence vector SV. It is designed to promote optimal element arrangement in the offspring while maintaining diversity.

The algorithm begins by creating a new offspring variable, Off, which represents the structure of the resulting offspring. The offspring can be initially empty or may contain default values. This critical initialization is performed in line 3 of the algorithm.

Next, the sequence vector SV's element e is iterated over by the method. This iterative loop structure, shown in line 4, demonstrates the algorithm's flexibility in handling a variety of input sequences. To enhance diversification, $n/10$ swap operations are applied to the solution vector SV. The algorithm then computes a probability vector, PV, for each element e. A normalizing procedure is used to the matching line of the matrix model M in order to generate this vector. This normalization makes sure the vector PV has a proper probability distribution since all of its values must add up to 1.

Next, from the probability vector PV, we sample a position p. This is done using a roulette wheel mechanism, where each index in PV is chosen according to its assigned probability. In the subsequent step, the algorithm assigns the element e to the sampled position p within the new individual. This operation holds the key to determining the precise placement of the element within the offspring structure.

Simultaneously, the algorithm ensures the dynamic update of the matrix model M. More specifically, it enforces a value of 0 at the position p that was sampled within M. This dynamic updating approach ensures that once a position is selected in one iteration, it cannot be chosen again in subsequent iterations. This mechanism upholds the crucial requirement of mutual exclusivity, a fundamental characteristic of permutations.

The algorithm iterates over all elements in the sequence vector SV, repeatedly sampling, assigning, and updating each element. This ensures that all elements are processed comprehensively. The iterative loop continues until all elements have been addressed. Upon completing the iteration, The algorithm produces the resulting offspring, denoted as Off, where nodes from the sequence vector are organized according to the positions sampled during the process.

To sum up, PGS-EDA uses probabilistic assignments given by the matrix model M to efficiently assign items from the sequence vector SV to locations inside the offspring Off. Positions are sampled, and the matrix model is updated dynamically throughout this process to guarantee that every element is positioned uniquely within its offspring while respecting the probability distribution defined by M.

5.3.4 Population update Strategy

The replacement strategy in an evolutionary algorithm plays a crucial role in determining how offspring are integrated into the current population.

We propose a replacement strategy in this context where an offspring becomes part of the population if its fitness is greater than the fitness of the weakest individual in the current generation. This mechanism guarantees that only the most highly suited solutions are integrated into the new generation, ultimately improving its overall quality as evolution progresses. Additionally, it ensures that no identical individuals are introduced, preserving diversity within the population.

This strategy effectively balances exploitation and exploration. It gradually replaces less fit individuals with stronger ones, propelling the population's evolution towards good solutions.

5.4 Experiments and Results

5.4.1 Experiments

To demonstrate the efficacy of our proposed algorithm, we conducted a series of experiments using a comprehensive set of 45 Permutation Flow-Shop Scheduling Problem (PFSP) instances obtained from Taillard's benchmark [89]. These selected instances were organized into nine distinct sets based on their size, which is defined by the number of jobs and the number of machines. The sets are as follows:

Jobs	Machines
20	5
20	10
20	20
50	5
50	10
50	20
100	5
100	10
100	20

Table 5.1: Selected Instances Organized by Size

For each size category, we specifically chose the first five instances for analysis.

In this study, we conducted a comparative analysis of our proposed method against some existing pure EDAs developed for permutation problems. These included Univariate Marginal Distribution (UMDA)[36], Edge Histogram-Based Sampling Algorithm (EHBSA)[44], Random-Key Estimation of Distribution Algorithm (RK-EDA)[38], Generalized Mallows EDAs (GM-EDA)[56], and Node Histogram-Based Sampling Algorithm (NHBSA) [48].

Our comparative study includes exclusively pure Estimation of Distribution Algorithms (EDAs), excluding any hybrid algorithms like HGM-EDA [56]. Our implementation covers PGS-EDA, EHBSA, NHBSA, and UMDA in Python, while the GM-EDA and RK-EDA codes are publicly available on the authors' GitHub repositories. The experiment parameters, detailed in Table 5.2, apply consistently to all algorithms. We determined the stopping criterion as $1000n^2$ fitness evaluations, a common choice across many studies [55, 38, 34].

Additionally, for PGS-EDA, ε was experimentally set to 0.002, and a total of $n/10$ inter-

Table 5.2: The algorithm's parameters.

Parameter	**Value**
Population size	$10n$
Selection size	n
Termination condition	$1000n^2$ FEs
Number of runs	10

change operations between two elements in SV were performed before the sampling phase. These parameters were chosen to ensure a fair and uniform experimental setup.

Throughout the comparative analysis of the various algorithms, we adhered to the parameters recommended by their respective authors. An important consideration is that both NHBSA and EHBSA utilized a sampling approach without template. The parameters for RK-EDA were directly adopted from the author's guidelines in [38]. It is worth mentioning that a variety of EDA variants employ models based on probability distance rankings, which can vary depending on the chosen distance function. In our evaluation, we specifically selected GM-EDAs employing the Kendall-τ distance [62, 108].

In our experiments, every algorithm under consideration was executed 10 times for each instance to ensure the robustness and reliability of the results. To evaluate their performance, we employed the Average Relative Percentage Deviation (ARPD) metric, which is defined as follows:

$$ARPD = \frac{1}{10}\sum_{i=1}^{10}\left(\frac{Found_i - Best}{Best} * 100\right) \tag{5.6}$$

Here, $Found_i$ represents the result obtained by the method in the i-th run, and $Best$ denotes the best-known value for the respective instance used in the experiment. ARPD allows us to quantify the algorithm's performance relative to the best-known solution, with lower ARPD values indicating better performance. This metric helps provide a standardized measure for comparing different algorithms across various instances and scenarios.

5.4.2 Results and Discussion

For a more comprehensive understanding of algorithm behavior, we categorized the Taillard benchmark instances into three distinct groups: small instances comprising 20 jobs, medium instances involving 50 jobs, and large instances featuring 100 jobs, all across 5, 10, and 20 machines.

Bayesian Performance Analysis (BPA) [109] is a powerful statistical tool for rigorously evaluating experimental results and uncovering the associated uncertainties. It plays a pivotal role in understanding the comparative performance of different algorithms.

The first critical step in the BPA process is to translate raw results into rankings for each specific test instance. This provides a structured basis for comparison. Next, BPA leverages the probabilistic realm by drawing samples from the posterior probability distribution of the Plackett-Luce model. This model is meticulously designed for handling rankings, making it an ideal choice for BPA. These drawn samples encompass the weights assigned to the algorithms and reflect the possible scenarios. The sampled weights are then used to assess the probability of each algorithm achieving success. In other words, BPA generates a probabilistic measure that quantifies each algorithm's likelihood of outperforming others. This procedure results in a thorough and detailed comprehension of the comparative effectiveness of the algorithms being examined, taking into account both the ranking of results and the corresponding uncertainty.[61, 109].

The ARPD outcomes for individual algorithms across the selected 20 job instances are detailed in Table 5.3. Notably, the most impressive results, highlighted in bold, underscore our proposed algorithm's exceptional performance. Specifically, our algorithm secures the top position for the minimum ARPD value in an impressive 11 out of 15 small instances considered. The next most successful contender, GM-EDA, outperforms other methods in approximately 13.33% of instances, closely followed by RK-EDA and NHBSA, each excelling in 6.33% of cases. Notably, the UMDA also delivers commendable results, especially when compared to the EHBSA.

To provide a more comprehensive overview, we calculated the mean ARPD value across all 15 cases for each method. In this regard, PGS-EDA emerges as a standout performer, showcasing its consistent effectiveness by maintaining the lowest average value. This outcome is a strong indicator of PGS-EDA's remarkable ability to consistently generate high-quality solutions across diverse instances.

Table 5.3: The ARPD (Average Relative Deviation Percentage) values obtained for instances involving 20 jobs and different configurations of 5, 10, and 20 machines.

Machines	**Instance**	**PGS-EDA**	**RK-EDA**	**NHBSA**	**EHBSA**	**GM-EDA**	**UMDA**
5	1	1.486	0.310	1.486	1.267	**0.000**	1.502
5	2	**0.250**	0.470	0.423	0.4415	0.412	1.530
5	3	1.267	**0.832**	1.794	3.977	1.378	3.5426
5	4	**0.699**	1.067	**0.699**	2.088	0.765	1.297
5	5	**0.622**	0.800	1.116	1.197	**0.622**	1.262
10	1	**0.982**	2.243	1.150	3.893	1.245	2.863
10	2	**1.048**	1.952	1.115	4.110	1.326	3.640
10	3	**1.537**	2.152	1.744	5.41	1.684	5.404
10	4	1.328	1.494	1.531	4.208	**1.291**	2.818
10	5	**1.279**	1.458	1.747	5.454	1.797	4.756
20	1	**0.959**	2.137	1.406	2.873	1.684	3.854
20	2	**1.083**	1.623	1.471	3.200	1.666	3.352
20	3	**0.741**	1.874	1.216	3.723	1.818	2.665
20	4	**0.890**	1.363	1.174	3.193	1.529	2.744
20	5	**0.816**	1.379	1.322	3.963	1.580	2.385
Mean		**1.097**	1.410	2.984	3.260	1.253	2.907

Figure 5.1 presents a visual representation of the posterior probability distribution, indicating the likelihood of each algorithm taking the top position. The black dots represent the observed empirical probabilities. Notably, PGS-EDA consistently exhibits the highest probability of achieving the top position, frequently reaching around 0.8. Following the Bayesian analysis results, NHBSA emerges as the second-strongest contender, with GM-EDA and RK-EDA closely behind, each with probabilities hovering around 0.3.

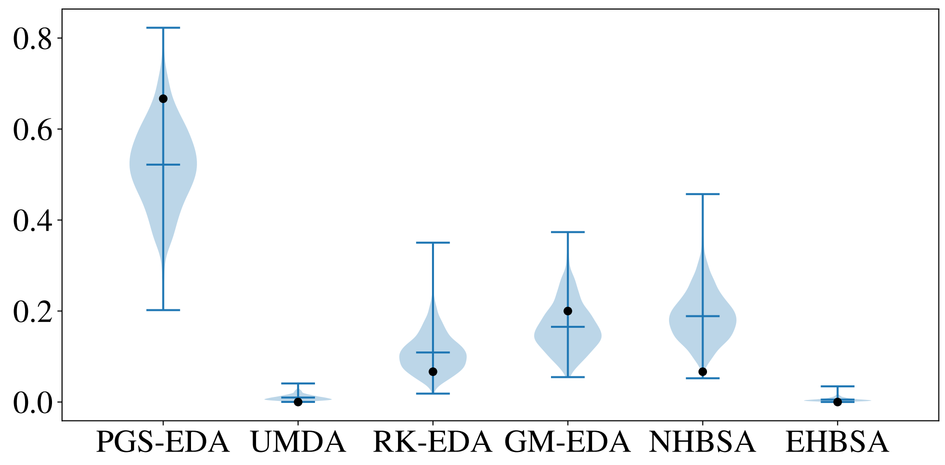

Figure 5.1: The Probability of the Considered Algorithms Achieving the Best Performance on Instances with 20 Jobs

Table 5.4 provides a detailed overview of ARPD results across instances with 50 jobs, with the most favorable ARPD values highlighted in bold. PGS-EDA emerges as the top-performing algorithm, consistently achieving the lowest ARPD values in nine instances. RK-EDA follows as the second most successful algorithm, outperforming others in four instances. GM-EDA secures the third spot, delivering strong results across multiple cases. UMDA and NHBSA also demonstrate favorable performance relative to EHBSA.

The precision of node (job) positioning significantly influences the makespan in the Permutation Flow-Shop Scheduling Problem (PFSP). The success of algorithms like PGS-EDA, RK-EDA, UMDA, and NHBSA can be attributed to their adept utilization of node-position sampling techniques, effectively optimizing solutions for specific instances.

Table 5.4: The ARPD (Average Relative Deviation Percentage) values obtained for instances involving 50 jobs and different configurations of 5, 10, and 20 machines.

Machines	**Instance**	**PGS-EDA**	**RK-EDA**	**NHBSA**	**EHBSA**	**GM-EDA**	**UMDA**
5	1	**0.000**	0.0954	0.055	1.189	0.359	0.220
5	2	0.656	**0.197**	0.366	2.194	0.688	2.183
5	3	0.347	0.148	**0.076**	1.900	0.209	0.763
5	4	**0.308**	0.843	0.367	2.988	0.654	1.768
5	5	**0.034**	**0.034**	**0.034**	1.847	**0.034**	0.838
10	1	2.466	3.246	3.365	8.710	**1.355**	3.295
10	2	1.943	1.894	3.692	9.576	**1.804**	3.613
10	3	2.263	**1.662**	3.509	10.391	1.710	3.690
10	4	**1.109**	1.354	1.409	9.839	1.174	3.253
10	5	**1.781**	1.791	2.106	10.521	1.791	4.377
20	1	**1.762**	1.809	3.788	10.699	2.918	3.669
20	2	**2.566**	3.343	4.411	11.551	4.209	5.289
20	3	**2.390**	3.189	5.697	12.108	3.966	4.929
20	4	1.820	**1.755**	3.336	10.621	2.494	3.587
20	5	2.417	**1.756**	2.486	9.982	2.825	4.709
Mean		1.456	1.539	2.213	7.607	1.754	3.078

Our method often performs well within various instances, exceeding other methods in 8 out of 15 cases and effectively minimizing makespan. Remarkably, it attains optimal solutions across all 10 runs for the 50×5 instance, which has an ARPD of 0.000. This highlights its superiority in scenarios with fewer machines. Meanwhile, the RK-EDA algorithm outperforms ours in 4 instances.

PGS-EDA outperforms all other algorithms on average, with the lowest mean ARPD value of 1.456 across the 50 job instances. RK-EDA, GM-EDA, NHBSA, and UMDA follow with mean ARPD values of 1.539, 1.754, 2.213, and 3.078, respectively. EHBSA has the highest mean ARPD value at 7.607, indicating that it may struggle to find near-optimal solutions on average.

Figure 5.2 shows the posterior distribution of probabilities for each algorithm, specifically focusing on their likelihood of being the top-performing solution. The black dots scattered across the chart represent the empirical probabilities for these algorithms.

PGS-EDA consistently emerges as the best, with the highest probability of obtaining the top position, sometimes exceeding 70%. On average, it maintains an impressive probability of around 0.4, reaffirming its superior performance across various problem instances. RK-EDA and GM-EDA closely follow PGS-EDA, occupying the second and third positions, respectively. Their probabilities are around 0.3 and 0.2. Their competitive performance further highlights the dynamic nature of the field, with these three algorithms consistently outperforming the rest in the pursuit of optimal solutions.

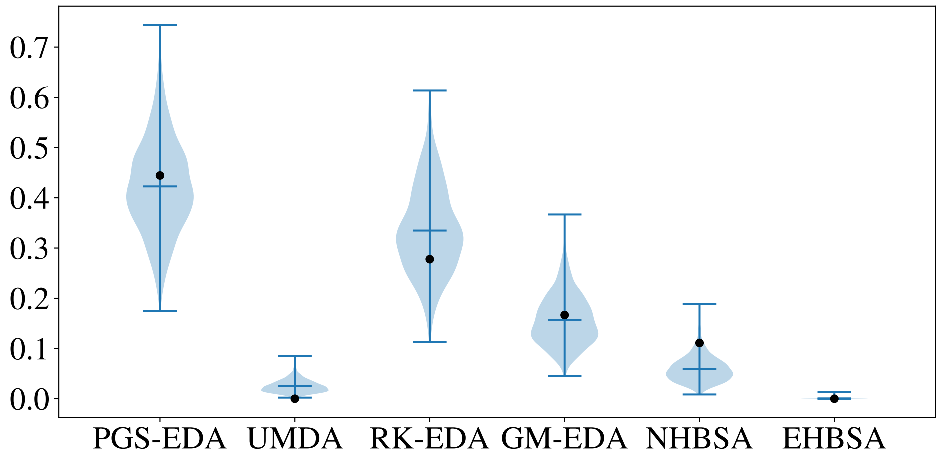

Figure 5.2: The Probability of the Considered Algorithms Achieving the Best Performance on Instances with 50 Jobs

The benchmark results for Taillard's 100-job instances are presented in Table 5.5. Notably, in instances with 5 machines, ARPD values for PGS-EDA, RK-EDA, GM-EDA, and NHBSA exhibit minimal variation. Specifically, RK-EDA achieves the lowest ARPD value in three out of five instances, while PGS-EDA and NHBSA each secure one instance with the minimum value.

This consistency suggests that, with a small number of machines, there is no statistically significant disparity among these top four algorithms. In essence, the performance differences between these algorithms could reasonably be attributed to random variations, and there isn't compelling evidence to assert the superiority of one algorithm over another in this particular scenario.

Table 5.5: The ARPD (Average Relative Deviation Percentage) values obtained for instances involving 100 jobs and different configurations of 5, 10, and 20 machines.

Machines	**Instance**	**PGS-EDA**	**RK-EDA**	**NHBSA**	**EHBSA**	**GM-EDA**	**UMDA**
5	1	0.032	**0.014**	0.028	0.709	0.032	0.190
5	2	**0.288**	0.398	0.360	1.319	0.335	0.598
5	3	0.206	**0.102**	0.502	2.035	0.372	1.023
5	4	0.165	**0.155**	0.197	1.804	0.241	0.597
5	5	0.059	0.057	**0.047**	2.066	0.085	0.604
10	1	1.018	**0.519**	1.057	7.383	1.019	4.523
10	2	1.009	0.736	1.916	8.104	**0.646**	3.047
10	3	0.982	0.246	1.409	6.059	**0.077**	5.512
10	4	2.469	**0.694**	2.538	7.572	1.179	5.289
10	5	1.309	**0.554**	1.877	8.247	0.846	4.322
20	1	2.816	**1.749**	4.740	11.191	2.083	7.397
20	2	2.371	**0.903**	4.718	12.770	1.560	6.851
20	3	4.710	**0.848**	4.408	11.376	1.788	8.523
20	4	3.843	**0.998**	3.659	11.718	1.665	7.980
20	5	4.861	**1.655**	4.496	10.749	2.005	8.659
Mean		1.742	**0.641**	2.130	6.874	0.929	4.341

Within the subgroup of cases involving 10 machines, we see a minor decline in the efficiency of PGS-EDA and NHBSA when compared to RK-EDA and GM-EDA. It is important to mention that despite this decline, PGS-EDA and NHBSA continue to be effective concerning ARPD. Within this particular situation, RK-EDA demonstrates superior performance compared to the other approaches in three out of the five occurrences, highlighting its usefulness. GM-EDA demonstrates superior performance by achieving the lowest ARPD among the other two examples.

However, as the number of machines increases, RK-EDA takes the lead by outperforming all other algorithms, securing the minimum ARPD in all five instances of 100×20. GM-EDA follows as the second most successful algorithm, surpassing PGS-EDA and NHBSA in this context. When considering the average ARPD value, PGS-EDA ranks third with a value of 1.471, while RK-EDA and GM-EDA attain significantly lower values of 0.624 and 0.954, respectively. These research results underscore the need to take into account the particular characteristics of a problem and the possible influence of variable parameters, such as the number of machines, while assessing and choosing suitable methods to tackle permutation problems.

The outcomes of the Bayesian Performance Analysis (BPA) for 100-job instances, indicating the likelihood of each algorithm being the top performer, are depicted in Figure 5.3. These results suggest that RK-EDA has the highest probability of emerging as the top-performing algorithm, with an empirical probability of approximately 0.8. In a closely trailing position, we find GM-EDA and PSG-EDA, both boasting a probability of around 0.4.

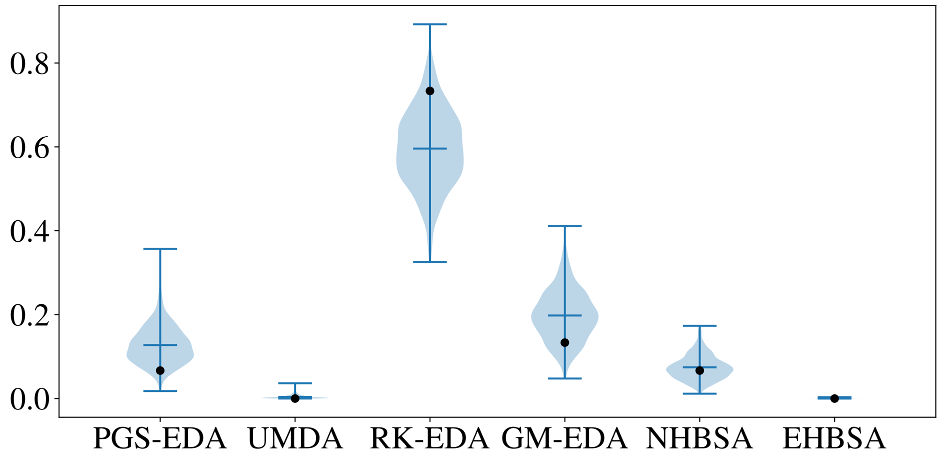

Figure 5.3: The Probability of the Considered Algorithms Achieving the Best Performance on Instances with 100 Jobs

In summary of our statistical analysis, a Bayesian performance analysis was carried out to assess the performance of all the algorithms on instances featuring 20, 50, and 100 jobs. The findings, as presented in Figure 5.4, indicate that PGS-EDA is the front-runner, possessing a probability of victory surpassing 50%. Following closely, we have RK-EDA as the second-best algorithm, with GM-EDA and NHBSA occupying the subsequent positions. The rest of the algorithms exhibit lower probabilities, all falling below the 0.05% mark.

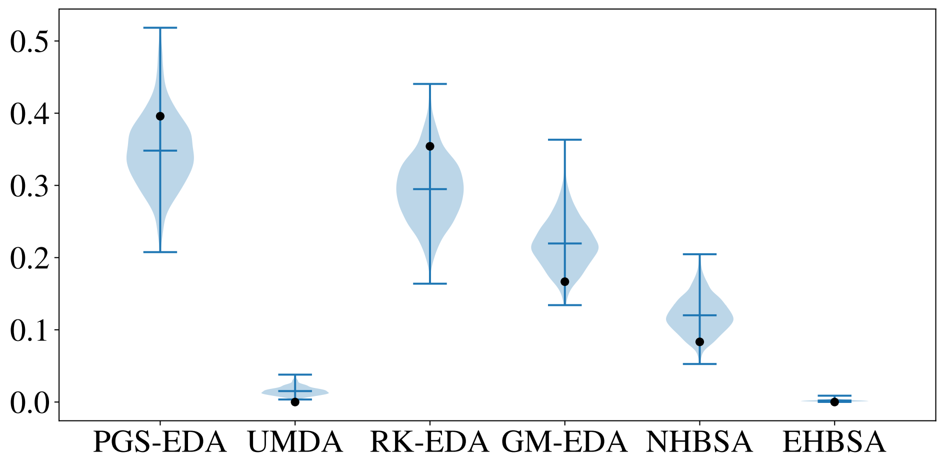

Figure 5.4: The Probability of the Considered Algorithms Achieving the Best Performance on all Selected Instances.

Figure 5.5 offers helpful information into the probabilities of each method getting positions between the top two and top three algorithms, as evaluated via Bayesian performance analysis across all instances featuring all instances of 20, 50, and 100 jobs. These results effectively reinforce our earlier observations, highlighting the robust performance of PGS-EDA and RK-EDA, which emerge as the most probable contenders for the top two positions. This further reinforces their effectiveness in addressing problems of this nature. Moreover, when expanding the scope to the top three algorithms, PGS-EDA retains its high confidence level, with RK-EDA and GM-EDA following closely behind. It's worth noting that NHBSA also demonstrates a noteworthy probability of approximately 0.4 for securing a place among the top three algorithms, further emphasizing its competitiveness in this context.

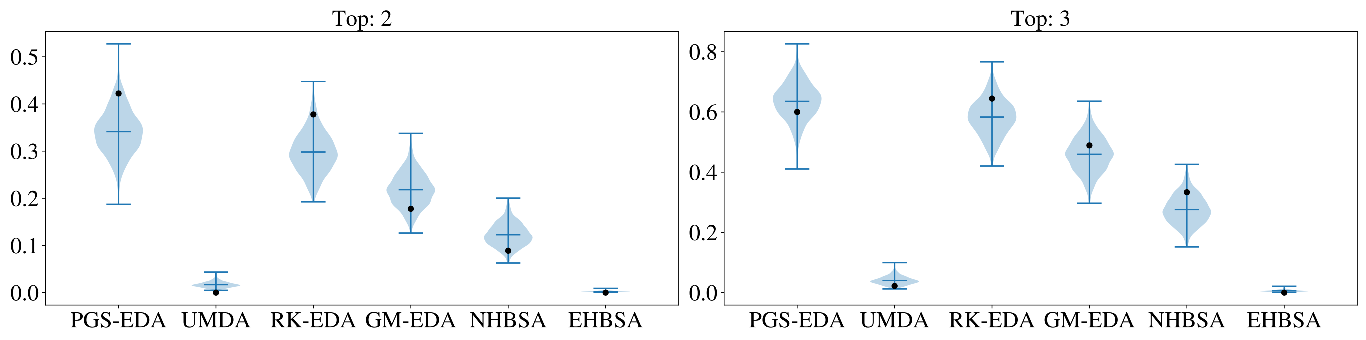

Figure 5.5: The probability of every algorithm obtaining a position within the top two and top three rankings

Bayesian performance analysis of the considered algorithms on permutation-based problems with 20, 50, and 100 jobs reveals that PGS-EDA has the highest probability of being the top performer, followed by RK-EDA. This supports the authors' claim that RK-EDA is well-suited for large-scale problems. GM-EDA and NHBSA also have competitive probabilities of being ranked in the top three. The other algorithms have lower probabilities of being ranked in the top positions. These results highlight the importance of algorithm selection based on the specific problem characteristics and parameter variations and the effectiveness of PGS-EDA and RK-EDA for addressing permutation-based problems.

Regarding processing time, GM-EDA demonstrates superior performance compared to all other algorithms, including PGS-EDA. This disparity in performance can be attributed to the efficiency of GM-EDA's implementation and its optimization strategies. Conversely, other algorithms, including PGS-EDA, share a comparable processing time, as they may utilize similar sampling processes or computational methodologies.

Although our approach has shown promising results in solving the Permutation Flow-Shop Scheduling Problem (PFSP), it is crucial to recognize the limitations of our work. The efficacy of our method may diminish with larger problem sizes, and broadening our research to encompass more types of problem situations would enhance the universality of our conclusions. Moreover, it is important to acknowledge that the effectiveness of PGS-EDA can alter when utilized in other problem categories or distinct problem size.

5.5 Conclusion

In summary, we've presented an Estimation of Distribution Algorithm (EDA) tailored for permutation problems. Our EDA employs a unique approach that combines matrix modeling and sequence vectors to allocate positions to nodes intelligently. This strategy utilizes probability information from the matrix model and element weight rankings in the sequence vector for effective position assignment.

The effectiveness of our EDA was thoroughly tested using Taillard's benchmark examples for the Permutation Flow-Shop Scheduling Problem (PFSP). Our findings provide clear proof of the success of our suggested algorithm in tackling the PFSP. The results we obtained on

Taillard’s benchmark instances show that our algorithm performs comparably to state-of-the-art methods when it comes to the quality of the solutions it produces. By clearly demonstrating its amazing skill in properly exploring and exploiting the solution space, our algorithm either surpassed current state-of-the-art solutions or achieved equivalent results.

We recommend several avenues for further research to overcome the limitations of our proposal and guide future investigations. One compelling direction involves conducting comparative studies with a variety of state-of-the-art algorithms, addressing a diverse set of permutation problems, including the Traveling Salesman Problem (TSP), Quadratic Assignment Problem (QAP), and Linear Ordering Problem (LOP). This would result in a more complete and reliable evaluation of our method’s performance.

Moreover, an exciting prospect lies in developing hybrid algorithms that integrate local search methods. This integration has the potential to enhance both the performance and scalability of our algorithm significantly. Techniques like Variable Neighborhood Search (VNS) could be explored to augment our algorithm’s capabilities further.

Chapter 6

General Conclusion

This chapter summarizes the key contributions and conclusions of the thesis and offers general directions for future research. Additionally, a list of resulting publications is provided.

6.1 Conclusions

This thesis explores the potential of novel algorithms based on Estimation of Distribution Algorithms (EDAs) and Generative Adversarial Networks (GANs) for solving permutation problems. Permutation problems are a class of optimization challenges with diverse real-world applications, such as scheduling, routing, and assignment problems. In the last years, GANs have great success in mimicking data distribution of continuous and discrete domains, but not in permutations, and it has not been investigated in permutation space.

In Chapter 2, we offer a detailed overview of permutation problems and EDAs. We discuss the unique characteristics of permutation problems and their complexities. We also introduce EDAs as a powerful class of evolutionary algorithms that are well-suited for solving many optimization problems. We present a taxonomy of EDAs and discuss specific variants that have been tailored for permutation problems. The chapter also highlighted the potential of generative deep learning models, especially Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs).

In Chapter 3, we introduce GAN-EDA, a novel EDA-based GAN specifically designed for solving permutation problems. GAN-EDA harnesses the power of GANs to learn a latent probabilistic model of selected individuals, which is then used to generate new individuals. We also introduce a novel permutation representation using one-hot matrices. We also implement a hybrid version of GAN-EDA that incorporates a 2-opt local search algorithm. Experimental results on the Traveling Salesman Problem (TSP) and Permutation Flow Shop Scheduling Problem (PFSP) benchmarks demonstrate that GAN-EDA achieves good results.

In Chapter 4, we introduce PGS-EDA, a pure EDA-based algorithm for solving the PFSP. PGS-EDA distinguishes itself through its novel sampling approach, which focuses on sampling positions rather than elements to generate new individuals. This innovative technique demonstrates impressive performance, especially for small and medium instance sizes, when compared to existing state-of-the-art algorithms.

Overall, our work has demonstrated the promise of EDAs and GANs for solving permutation problems. We believe that our proposed algorithms can be used as a foundation for developing even more effective and efficient algorithms for solving this important class of optimization problems.

In addition to the contributions mentioned above, our work has also shed light on the following aspects of using EDAs and GANs for solving permutation problems:

- EDAs that are specifically designed for permutation problems are more effective than EDAs that are generalized from continuous or discrete domains.

- Our research highlights that there is no universally superior algorithm that consistently outperforms state-of-the-art methods across all permutation problems. This means different problems may require tailored algorithms or combinations of techniques to achieve optimal results.

- EDAs and GANs can be used to learn complex relationships between the different components of a permutation.

- Alternative permutation representations, such as one-hot matrices, can improve the performance of EDAs and GANs on permutation problems.

- Hybrid algorithms that combine EDAs with local search algorithms can further improve the performance of EDAs on permutation problems.

6.2 Future work

This thesis has explored the potential of estimation of distribution algorithms (EDAs) and generative adversarial networks (GANs) for solving permutation problems. While our proposed algorithms have achieved promising results, there are still many opportunities for further exploration and improvement.

Here are some specific future directions that we believe have the potential to lead to significant advancements:

- **Diversifying GAN variants:** Our current GAN-EDA algorithm uses vanilla GANs. However, many other GAN variants have been developed in recent years, each with its strengths and weaknesses. Exploring the use of alternative GAN variants in EDA-based algorithms could lead to significant performance improvements.

- **Incorporating variational autoencoders (VAEs):** VAEs are another type of generative model that has shown great promise for a variety of tasks. VAEs can learn complex latent representations of data, which could be beneficial for solving permutation problems. Investigating how VAEs can be incorporated into EDA-based algorithms is a promising area for future research.

- **Expanding to other combinatorial problems:** EDAs and GANs are not limited to permutation problems. In principle, they can be applied to a wide range of combinatorial optimization problems. We encourage researchers to explore the use of EDAs and GANs for solving other types of combinatorial problems, such as quadratic assignment problems and vehicle routing problems.

- **Solving real-world problems:** Permutation problems arise in a wide range of real-world applications. By exploring how EDAs and GANs can be harnessed to solve real-world problems involving permutations, we can pave the way for practical solutions in areas such as scheduling, logistics, and data analysis.

- **Hybridizing with local search:** Local search algorithms such as VNS are powerful tools for solving combinatorial optimization problems. In our cases, hybridizing PGS-EDA with local search algorithms can lead to significant performance improvements, especially for large instances.

We believe that the future of EDA-based algorithms and GANs for solving permutation problems is very bright. By exploring the directions outlined above, we can make significant progress toward developing more effective and efficient algorithms for solving this important class of optimization problems.

6.3 Publications

During the period of this research, a number of research publications were published that contributed to certain chapters of the thesis. The publications are listed as follows :

6.3.1 Referred journals

- Sami Lemtenneche, Abdallah Bensayah, and Abdelhakim Cheriet. An estimation of distribution algorithm for permutation flow-shop scheduling problem. Systems, 11(8):389, 2023.

6.3.2 Conference communications

- Sami Lemtenneche, Abdelhakim Cheriet, and Bensayah Abdellah. Permutation-based optimization using a generative adversarial network. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO ’21, page 159–160, New York, NY, USA, 2021. Association for Computing Machinery

- Sami Lemtenneche, Abdelhakim Cheriet, and Abdallah Bensayah. Introducing generative adversarial networks on estimation of distribution algorithm to solve permutation-based problems. In 2022 International Symposium on iNnovative Informatics of Biskra (ISNIB), pages 1–5, 2022.

Bibliography

[1] Mark Hauschild and Martin Pelikan. An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation*, 1(3):111–128, 2011.

[2] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Mathematical Sciences Series. W. H. Freeman, 1979.

[3] Ran Cheng, Cheng He, Yaochu Jin, and Xin Yao. Model-based evolutionary algorithms: a short survey. *Complex & Intelligent Systems*, 4, 08 2018.

[4] Rocio Vargas, Amir Mosavi, and Ramon Ruiz. Deep learning: a review. 2017.

[5] Josu Ceberio. *Solving permutation problems with estimation of distribution algorithms and extensions thereof*. PhD thesis, PhD thesis, Universidad del País Vasco-Euskal Herriko Unibertsitatea, 2014.

[6] David E Goldberg and Robert Lingle. Alleles, loci, and the traveling salesman problem. In *Proceedings of the first international conference on genetic algorithms and their applications*, pages 154–159. Psychology Press, 2014.

[7] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[8] Tjalling C. Koopmans and Martin Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25(1):53–76, 1957.

[9] A. B. Chandramouli. Heuristic approach for n-job, 3-machine flow shop scheduling problem involving transportation time, break down time and weights of jobs. *Mathematical and Computational Applications*, 10(2):301–305, 2005.

[10] Huan Liu, Fuqing Zhao, Ling Wang, Jie Cao, Jianxin Tang, and Jonrinaldi. An estimation of distribution algorithm with multiple intensification strategies for two-stage hybrid flow-shop scheduling problem with sequence-dependent setup time. *Applied Intelligence*, 53(5):5160–5178, 2023.

[11] Tingyi Zhang, Xiaopeng Guo, and Guojun Ji. Permutation flow shop scheduling optimization method based on cooperative games. *IEEE Access*, 11:47377–47389, 2023.

[12] Mehdi Foumani, Ayaz Razeghi, and Kate Smith-Miles. Stochastic optimization of two-machine flow shop robotic cells with controllable inspection times: From theory toward practice. *Robotics and Computer-Integrated Manufacturing*, 61:101822, 2020.

[13] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[14] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[15] Rubén Armañanzas, Iñaki Inza, Roberto Santana, Yvan Saeys, Jose Luis Flores, Jose Lozano, Yves Van de Peer, Rosa Blanco, Víctor Robles, Concha Bielza, and Pedro Larranaga. A review of estimation of distribution algorithms in bioinformatics. *BioData mining*, 1:6, 10 2008.

[16] H Mühlenbein and G Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature — PPSN IV*, pages 178–187, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[17] Pedro Larranaga Jose A. Lozano. *ESTIMATION OF DISTRIBUTION ALGORITHMS: A New Tool for Evolutionary Computation*. Springer Sciencc+Business Media New York, 2002.

[18] Martin Pelikan Mark Hauschild. An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation*, 2011.

[19] Iñaki Inza Endika Bengoetxea Jose A. Lozano, Pedro Larrañaga. *Towards a new evolutionary computation: advances on estimation of distribution algorithms*, volume 192. Springer Science & Business Media, 2006.

[20] Shumeet Baluja. *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning*. School of Computer Science, Carnegie Mellon University Pittsburgh, PA, 1994.

[21] Georges R Harik, Fernando G Lobo, and David E Goldberg. The compact genetic algorithm. *IEEE transactions on evolutionary computation*, 3(4):287–297, 1999.

[22] Jeremy De Bonet, Charles Isbell, and Paul Viola. Mimic: Finding optima by estimating probability densities. *Advances in neural information processing systems*, 9, 1996.

[23] Shumeet Baluja and Scott Davies. Using optimal dependency-trees for combinational optimization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 30–38, 1997.

[24] Martin Pelikan and Heinz Mühlenbein. The bivariate marginal distribution algorithm. In *Advances in soft computing: Engineering design and manufacturing*, pages 521–535. Springer, 1999.

[25] Martin Pelikan, David E Goldberg, Erick Cantú-Paz, et al. Boa: The bayesian optimization algorithm. In *Proceedings of the genetic and evolutionary computation conference GECCO-99*, volume 1. Citeseer, 1999.

[26] Ramon Etxeberria. Global optimization using bayesian networks. In *Proc. 2nd Symposium on Artificial Intelligence (CIMAF-99)*, 1999.

[27] Siddhartha Shakya and Roberto Santana. An eda based on local markov property and gibbs sampling. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 475–476, 2008.

[28] Roberto Santana. A markov network based factorized distribution algorithm for optimization. In *Machine Learning: ECML 2003: 14th European Conference on Machine*

Learning, Cavtat-Dubrovnik, Croatia, September 22-26, 2003. Proceedings 14, pages 337–348. Springer, 2003.

[29] Roberto Santana. Estimation of distribution algorithms with kikuchi approximations. *Evolutionary Computation*, 13(1):67–97, 2005.

[30] Georges R Harik, Fernando G Lobo, and Kumara Sastry. Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ecga). In *Scalable optimization via probabilistic modeling*, pages 39–61. Springer.

[31] Tian-Li Yu, David E Goldberg, Kumara Sastry, Claudio F Lima, and Martin Pelikan. Dependency structure matrix, genetic algorithms, and effective recombination. *Evolutionary computation*, 17(4):595–626, 2009.

[32] Martin Pelikan, Mark Hauschild, and Fernando G. Lobo. Estimation of distribution algorithms. In Janusz Kacprzyk and Witold Pedrycz, editors, *Springer Handbook of Computational Intelligence*, Springer Handbooks, pages 899–928. Springer, 2015.

[33] Alain Bertrand Bomgni, Miguel Landry Foko Sindjoung, Dhalil Kamdem Tchibonsou, Mthulisi Velempini, and Jean Frédéric Myoupo. Neseprin: A new scheme for energy-efficient permutation routing in iot networks. *Computer Networks*, 214:109162, 2022.

[34] Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and JoseA Lozano. A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. *Progress in Artificial Intelligence*, 1:103–117, 04 2012.

[35] Peter AN Bosman and Dirk Thierens. Expanding from discrete to continuous estimation of distribution algorithms: The id a. In *International Conference on Parallel Problem Solving from Nature*, pages 767–776. Springer, 2000.

[36] J. A. Lozano and A. Mendiburu. *Estimation of Distribution Algorithms Applied to the Job Shop Scheduling Problem: Some Preliminary Research*, pages 231–242. Springer US, Boston, MA, 2002.

[37] James C Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, 6(2):154–160, 1994.

[38] Mayowa Ayodele, John Mccall, and Olivier Regnier-Coudert. Rk-eda: A novel random key based estimation of distribution algorithm. volume 9921, pages 849–858, 09 2016.

[39] Peter AN Bosman and Dirk Thierens. Crossing the road to efficient ideas for permutation problems. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 219–226, 2001.

[40] Max Henrion. Propagating uncertainty in bayesian networks by probabilistic logic sampling. In *Machine intelligence and pattern recognition*, volume 5, pages 149–163. Elsevier, 1988.

[41] Martin Pelikan, Shigeyoshi Tsutsui, and Rajiv Kalapala. Dependency trees, permutations, and quadratic assignment problem. In *Genetic And Evolutionary Computation Conference: Proceedings of the 9 th annual conference on Genetic and evolutionary computation*, volume 7, pages 629–629. Citeseer, 2007.

[42] E Bengoetxea, P Larranaga, I Bloch, A Perchant, and C Boeres. Inexact graph matching using learning and simulation of bayesian networks. In *Proceedings of CaNew workshop, ECAI*, 2000.

[43] Optimization in continuous domains by learning and simulation of gaussian networks. In *Proc. of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, 2000.

[44] Shigeyoshi Tsutsui. Probabilistic model-building genetic algorithms in permutation representation domain using edge histogram. In Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, Hans-Paul Schwefel, and José-Luis Fernández-Villacañas, editors, *Parallel Problem Solving from Nature — PPSN VII*, pages 224–233, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[45] Shigeyoshi Tsutsui and Gordon Wilson. Solving capacitated vehicle routing problems using edge histogram based sampling algorithms. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, volume 1, pages 1150–1157. IEEE, 2004.

[46] Shigeyoshi Tsutsui and Mitsunori Miki. Using edge histogram models to solve flow shop scheduling problems with probabilistic model-building genetic algorithms. In *Recent Advances in Simulated Evolution and Learning*, pages 230–249. World Scientific, 2004.

[47] Shigeyoshi Tsutsui and Mitsunori Miki. Solving flow shop scheduling problems with probabilistic model-building genetic algorithms using edge histograms. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*, pages 465–471, 2002.

[48] S. Tsutsui. Node Histogram vs. Edge Histogram: A Comparison of Probabilistic Model-Building Genetic Algorithms in Permutation Domains. *2006 IEEE International Conference on Evolutionary Computation*, (2006009):1939–1946, 2006.

[49] Shigeyoshi Tsutsui. A comparative study of sampling methods in node histogram models with probabilistic model-building genetic algorithms. In *2006 IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 3132–3137. IEEE, 2006.

[50] Mayer Alvo and LH Philip. *Statistical methods for ranking data*, volume 1341. Springer, 2014.

[51] Robin L Plackett. The analysis of permutations. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 24(2):193–202, 1975.

[52] R Duncan Luce. Individual choice behavior. 1959.

[53] Josu Ceberio. The plackett-luce ranking model on permutation-based optimization problems. 06 2013.

[54] Colin L Mallows. Non-null ranking models. i. *Biometrika*, 44(1/2):114–130, 1957.

[55] Josu Ceberio, Alexander Mendiburu, and Jose Lozano. Introducing the mallows model on estimation of distribution algorithms. volume 7063, pages 461–470, 11 2011.

[56] Josu Ceberio, E. Irurozqui, Alexander Mendiburu, and Jose Lozano. A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 18, 04 2013.

[57] Etor Arza, Josu Ceberio, Aritz Pérez, and Ekhiñe Irurozki. Approaching the quadratic assignment problem with kernels of mallows models under the hamming distance. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 141–142, 2019.

[58] Ekhine Irurozki, Borja Calvo, and Jose A Lozano. Mallows model under the ulam distance: a feasible combinatorial approach. In *Workshop Book 2014 NIPS Conference, Montreal, Quebec, Canada*, 2014.

[59] Ekhine Irurozki, Borja Calvo, and Jose A Lozano. Sampling and learning mallows and generalized mallows models under the cayley distance. *Methodology and Computing in Applied Probability*, 20:1–35, 2018.

[60] Etor Arza, Aritz Perez, Ekhine Irurozki, and Josu Ceberio. Kernels of mallows models under the hamming distance for solving the quadratic assignment problem. *Swarm and Evolutionary Computation*, 59:100740, 2020.

[61] Josu Ceberio, Alexander Mendiburu, and Jose A. Lozano. Kernels of mallows models for solving permutation-based problems. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, page 505–512, New York, NY, USA, 2015. Association for Computing Machinery.

[62] Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and Jose A. Lozano. A review of distances for the mallows and generalized mallows estimation of distribution algorithms. *Comput. Optim. Appl.*, 62(2):545–564, nov 2015.

[63] Mayowa Ayodele, John McCall, Olivier Regnier-Coudert, and Liam Bowie. A random key based estimation of distribution algorithm for the permutation flowshop scheduling problem. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2364–2371, 2017.

[64] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 2021.

[65] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5(64-67):2, 2001.

[66] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[67] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2):26–31, 2012.

[68] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[69] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[70] Ruoqi Wei and Ausif Mahmood. Recent advances in variational autoencoders with representation learning for biomedical informatics: A survey. *Ieee Access*, 9:4939–4956, 2020.

[71] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[72] Ian J Goodfellow, Jean Pouget-abadie, Mehdi Mirza, Bing Xu, and David Warde-farley. Generative Adversarial Nets. pages 1–9, 2014.

[73] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[74] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[75] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, 29, 2016.

[76] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

[77] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

[78] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.

[79] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[80] Tzu-An Song, Samadrita Roy Chowdhury, Fan Yang, and Joyita Dutta. PET image super-resolution using generative adversarial networks. *Neural Networks*, 125:83–91, 2020.

[81] Sai Rajeswar, Sandeep Subramanian, Francis Dutil, Christopher Pal, and Aaron Courville. Adversarial generation of natural language. *arXiv preprint arXiv:1705.10929*, 2017.

[82] C. He, S. Huang, R. Cheng, K. C. Tan, and Y. Jin. Evolutionary multiobjective optimization driven by generative adversarial networks (gans). *IEEE Transactions on Cybernetics*, pages 1–14, 2020.

[83] Josu Ceberio, Alexander Mendiburu, and Jose Lozano. Kernels of mallows models for solving permutation-based problems. pages 505–512, 07 2015.

[84] S. Tsutsui, Martin Pelikan, and D.E. Goldberg. Using edge histogram models to solve permutation problems with probabilistic model-building genetic algorithms. *IlliGAL Report*, 2003022(2003022), 2003.

[85] Malte Probst. Generative adversarial networks in estimation of distribution algorithms for combinatorial optimization. 09 2015.

[86] Menghui Chen, Ruiran Yu, Shengjian Xu, Yifei Luo, and Zhihua Yu. An improved algorithm for solving scheduling problems by combining generative adversarial network with evolutionary algorithms. pages 1–7, 10 2019.

[87] Qingfu Zhang, Jianyong Sun, Edward Tsang, and John Ford. Estimation of distribution algorithm with 2-opt local search for the quadratic assignment problem. *Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms*, pages 281–292, 2006.

[88] Gerhard Reinelt. TSPLIB—A Traveling Salesman Problem Library. *INFORMS Journal on Computing*, 3(4):376–384, November 1991.

[89] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993. Project Management anf Scheduling.

[90] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[91] Lei Sun, Weimin Shi, Junru Wang, Huimin Mao, Jiajia Tu, and Luojun Wang. Research on production scheduling technology in knitting workshop based on improved genetic algorithm. *Applied Sciences*, 13(9), 2023.

[92] Bilal Khurshid, Shahid Maqsood, Muhammad Omair, Biswajit Sarkar, Imran Ahmad, and Khan Muhammad. An improved evolution strategy hybridization with simulated annealing for permutation flow shop scheduling problems. *IEEE Access*, 9:94505–94522, 2021.

[93] APG Brown and ZA Lomnicki. Some applications of the “branch-and-bound” algorithm to the machine scheduling problem. *Journal of the Operational Research Society*, 17(2):173–186, 1966.

[94] ZA Lomnicki. A “branch-and-bound” algorithm for the exact solution of the three-machine scheduling problem. *Journal of the operational research society*, 16:89–100, 1965.

[95] Muhammad Nawaz, E Emory Enscore Jr, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[96] Christos Koulamas. A new constructive heuristic for the flowshop scheduling problem. *European Journal of Operational Research*, 105(1):66–71, 1998.

[97] Pavol Semanco and Vladimir Modrak. A comparison of constructive heuristics with the objective of minimizing makespan in the flow-shop scheduling problem. *Acta Polytechnica Hungarica*, 9(5):177–190, 2012.

[98] Damla Kizilay, Mehmet Fatih Tasgetiren, Quan-Ke Pan, and Liang Gao. A variable block insertion heuristic for solving permutation flow shop scheduling problem with makespan criterion. *Algorithms*, 12(5), 2019.

[99] Vince Caraffa, Stefano Ianes, Tapan P Bagchi, and Chelliah Sriskandarajah. Minimizing makespan in a blocking flowshop using genetic algorithms. *International Journal of Production Economics*, 70(2):101–115, 2001.

[100] Mohamed Abdel-Basset, Reda Mohamed, Mohamed Abouhawwash, Ripon K. Chakrabortty, and Michael J. Ryan. A simple and effective approach for tackling the permutation flow shop scheduling problem. *Mathematics*, 9(3), 2021.

[101] Ibrahim H Osman and CN Potts. Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6):551–557, 1989.

[102] Ricardo Pérez-Rodríguez. A hybrid estimation of distribution algorithm for the quay crane scheduling problem. *Mathematical and Computational Applications*, 26(3), 2021.

[103] M. Fatih Tasgetiren, Yun-Chia Liang, Mehmet Sevkli, and Gunes Gencyilmaz. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 177(3):1930–1947, 2007.

[104] Iqbal Hayat, Adnan Tariq, Waseem Shahzad, Manzar Masud, Shahzad Ahmed, Muhammad Umair Ali, and Amad Zafar. Hybridization of particle swarm optimization with

variable neighborhood search and simulated annealing for improved handling of the permutation flow-shop scheduling problem. *Systems*, 11(5), 2023.

[105] Adil Baykasoğlu and Mümin Emre Şenol. Parallel wsar for solving permutation flow shop scheduling problem. *Computer Sciences amp; Mathematics Forum*, 2(1), 2022.

[106] Jonathan Huang, Carlos Guestrin, and Leonidas Guibas. Fourier theoretic probabilistic inference over permutations. *Journal of machine learning research*, 10(5), 2009.

[107] Bassem Jarboui, Mansour Eddaly, and Patrick Siarry. An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. *Computers & Operations Research*, 36(9):2638–2646, 2009.

[108] Ekhine Irurozki, Josu Ceberio, Josean Santamaria, Roberto Santana, and Alexander Mendiburu. Algorithm 989: Perm mateda: A matlab toolbox of estimation of distribution algorithms for permutation-based combinatorial optimization problems. *ACM Trans. Math. Softw.*, 44(4), jul 2018.

[109] Jairo Rojas-Delgado, Josu Ceberio, Borja Calvo, and Jose A. Lozano. Bayesian performance analysis for algorithm ranking comparison. *IEEE Transactions on Evolutionary Computation*, 26(6):1281–1292, 2022.