TU WIEN Informatics

Denoising Diffusion-Based Evolutionary Algorithms

Exploring Hybridizations of Evolutionary Algorithms with Denoising Diffusion Models

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Joan Salvà Soler
Matrikelnummer 12223411

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Günther Raidl, Ao.Univ.Prof. Dipl.-Ing. Dr.techn.

Wien, 31. März 2025

Joan Salvà Soler   Günther Raidl

TU WIEN Informatics

Denoising Diffusion-Based Evolutionary Algorithms

Exploring Hybridizations of Evolutionary Algorithms with Denoising Diffusion Models

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Joan Salvà Soler
Registration Number 12223411

to the Faculty of Informatics

at the TU Wien

Advisor: Günther Raidl, Ao.Univ.Prof. Dipl.-Ing. Dr.techn.

Vienna, March 31, 2025

Joan Salvà Soler

Günther Raidl

Erklärung zur Verfassung der Arbeit

Joan Salvà Soler

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 31. März 2025

Joan Salvà Soler

Acknowledgements

First, I would like to sincerely thank my supervisor, Günther, for his invaluable support, insightful ideas, and the time he dedicated to guiding me throughout this journey. His expertise has been key in shaping this work, and I truly appreciate his patience and dedication.

To my friends and roommates: It's been fun to write the thesis at the same time (with some of you) and to share the journey. And it's nice to hang out to keep having a life outside of so much work. To Eva, thanks for everything.

To my family—despite the distance and the fact that you might not always know exactly what I'm working on: my entire journey has been possible only because of your support.

Finally, thanks to the team in Quantagonia, for their support, understanding, and flexibility, which allowed me to successfully complete this thesis while working on amazing projects with them.

Kurzfassung

In dieser Arbeit wird die Integration von entrauschenden Diffusionsmodellen mit evolutionären Algorithmen (EAs) zur Verbesserung der maschinellen lernbasierten kombinatorischen Optimierung untersucht. Während Diffusionsmodelle vielversprechende generative Ansätze zur Lösung von NP-schweren Problemen bieten, sind sie häufig auf problemspezifische Heuristiken angewiesen und verfügen nicht über die robuste Explorationsfähigkeit klassischer Suchverfahren. Um diese Einschränkungen zu überwinden, schlagen wir einen neuartigen diffusionsbasierten evolutionären Algorithmus (DEA) vor. Dieser Lösungsanstz nutzt vortrainierte Diffusionsmodelle sowohl zur Initialisierung der Population als auch für die Rekombination, wodurch der durch Diffusionsmodelle erfasste multimodale Lösungsraum gezielt in die explorative Suchstruktur eines EAs eingebettet wird. Wir fokussieren uns auf das Maximum-Independent-Set-Problem (MIS) und erweitern einen Basis-EA um eine diffusionsgestützte Initialisierung sowie einen neuartigen Rekombinationsoperator. Letzterer wird mittels Imitationslernens aus einem optimalen Rekombinationsdemonstrator trainiert, der als Integer Linear Program formuliert ist. Numerische Experimente auf Erdős-Rényi-Graphen unterschiedlicher Größe zeigen, dass DEA die Leistung von Difusco, einem hochmodernen diffusionsbasierten Optimierer für kombinatorische Optimierungsprobleme, signifikant übertrifft, indem es sowohl eine höhere Lösungsqualität als auch eine bessere Generalisierungsfähigkeit auf größere Graphinstanzen erzielt. Insbesondere schlägt DEA die führende Software zur Lösung gemischt-ganzzahligen linearen Problemen, Gurobi, bei größeren Instanzen unter gleichen Zeitbudgets, bleibt jedoch hinter hochspezialisierten klassischen Lösern wie KaMIS zurück. Ablationsstudien belegen den entscheidenden Einfluss der diffusionsbasierten Initialisierung und Rekombination auf die Gesamtleistung des DEA-Frameworks. Diese Arbeit liefert empirische Evidenz für das synergetische Potenzial der Kombination von Diffusionsmodellen mit evolutionären Algorithmen und eröffnet neue Perspektiven für die Entwicklung robuster hybrider ML-metaheuristischer Ansätze zur Lösung komplexer kombinatorischer Optimierungsprobleme.

Abstract

This thesis explores the integration of denoising diffusion models with evolutionary algorithms (EAs) to enhance machine learning-based combinatorial optimization. While diffusion models have shown promise as generative solvers for NP-hard problems, they often rely on problem-specific heuristics and lack the robust exploration capabilities of traditional search methods. To address these limitations, we propose a novel Diffusion-based Evolutionary Algorithm (DEA) framework. This approach leverages pre-trained diffusion models for both population initialization and recombination, effectively exploiting the multimodal solution space captured by diffusion models within the exploratory structure of an EA. We focus on the Maximum Independent Set (MIS) problem, developing a baseline EA and then introducing diffusion-based initialization and a diffusion recombination operator trained via imitation learning from an optimal recombination demonstrator formulated as an Integer Linear Program. Computational experiments on Erdős-Rényi graph datasets of varying sizes demonstrate that DEA significantly outperforms standalone Difusco, a state-of-the-art diffusion-based solver for combinatorial optimization, achieving improved solution quality and better out-of-distribution generalization to larger graph instances. Notably, DEA outperforms the MIP solver Gurobi on larger instances when considering the same time limit, while still falling short of highly specialized classical solvers like KaMIS. Our ablation studies highlight the crucial contributions of both diffusion-based initialization and recombination to the DEA framework's success. This work provides empirical evidence for the synergistic potential of combining diffusion models and EAs, offering a promising direction for developing more robust and effective machine learning approaches to complex combinatorial optimization problems and paving the way for future research into hybrid ML-metaheuristic methods.

Contents

CHAPTER 1

Introduction

This chapter introduces the main concepts and objectives of this thesis. In Section 1.1, we present the core concepts and background necessary for understanding our work. In Section 1.2, we identify the research gaps and motivate our approach. Section 1.3 outlines the objectives and research questions that guide this thesis, along with the expected contributions. Finally, Section 1.4 provides an overview of the organization of the remaining chapters.

1.1 Background and Context

We start by introducing the main concepts for this work: Combinatorial Optimization (CO) and its challenges, common classical methods for CO, and machine learning for CO, including its limitations.

1.1.1 Combinatorial Optimization

Combinatorial Optimization (CO) is a branch of optimization that focuses on selecting the best solution from a finite, yet often exponentially large, set of possibilities [KV12]. At its core, CO is concerned with problems where one must optimize an objective function—such as minimizing cost or maximizing efficiency—subject to specific constraints. Many of these problems are classified as NP-hard, meaning that finding an exact solution quickly (in polynomial time) is generally infeasible for large-scale instances. This computational complexity has led researchers to develop a variety of approximate, heuristic, and metaheuristic approaches.

The importance of CO problems spans multiple disciplines and industries, among which are:

- **Logistics and Transportation**: Problems like the Traveling Salesperson Problem (TSP) [ABCC07] and Vehicle Routing Problem (VRP) [TV02] are classic examples of CO problems. Efficient routing and scheduling directly impact cost savings, delivery times, and overall operational efficiency.
- **Finance**: In portfolio optimization and asset allocation, CO methods are used to determine the best combination of investments to maximize returns while managing risk, considering a multitude of constraints [MOS15].
- **Telecommunications**: Network design, bandwidth allocation, and signal routing are optimized using CO techniques, ensuring robust and efficient communication networks [RP06].
- **Manufacturing and Production**: Scheduling tasks, resource allocation, and supply chain management benefit from CO by improving productivity and reducing operational costs [Cra97].
- **Computer Science and Operations Research**: From task scheduling in computing systems to resource allocation in cloud services, CO plays a critical role in ensuring systems perform optimally under given constraints [KAACA15].

1.1.2 NP-Hardness and Computational Challenges

A significant number of CO problems fall into the NP-hard category [GJ79], meaning that there is no known algorithm capable of solving all instances of these problems in polynomial time. This NP-hardness implies that as the size of the problem increases, the number of potential solutions grows exponentially, often making exact methods computationally infeasible for large-scale instances. For example, problems such as the Traveling Salesperson Problem (TSP) quickly become intractable as the number of elements increases [KV12].

Due to this inherent complexity, researchers have focused on developing approximate algorithms, heuristics, and metaheuristics that can provide high-quality solutions within reasonable computational times. These approaches trade off guaranteed optimality for practicality and efficiency [CGJ96]. In many practical applications, the slight loss in optimality is acceptable when balanced against significant reductions in computational time and resources.

1.1.3 Classical Methods for Combinatorial Optimization

Traditionally, CO problems have been addressed using three main classes of methods: exact algorithms, heuristics, and metaheuristics. Each approach has its strengths and limitations, shaped by the trade-off between solution quality and computational feasibility.

Exact Methods Exact methods, such as Integer Linear Programming (ILP) and Constraint Programming (CP), aim to find provably optimal solutions. ILP formulates problems using linear constraints and integer variables, often solved via branch-and-bound algorithms [Wol20]. CP, on the other hand, uses declarative constraints and domain filtering to systematically explore the solution space [RvBW06]. While these methods guarantee optimality, their computational complexity grows exponentially with problem size, making them impractical for large-scale NP-hard instances [GJ79].

Heuristic Methods Heuristic methods focus on finding good feasible solutions quickly, often sacrificing optimality for computational efficiency. Examples are construction heuristics, which use a greedy policy for sequentially building a feasible solution, and local search algorithms, which iteratively improve solutions by exploring neighboring configurations [AL03]. While heuristics are computationally efficient, they lack guarantees on solution quality and are often problem-specific, requiring tailored designs for each application.

Metaheuristics Metaheuristics provide high-level frameworks for guiding heuristic search, offering a balance between exploration and exploitation. Evolutionary algorithms (EAs), for instance, maintain a population of solutions that evolve over generations through selection, crossover, and mutation operators [ES15]. Other metaheuristics, such as Tabu Search [Glo89] and Ant Colony Optimization [DG97], use memory-based or population-based strategies to navigate complex solution spaces. While metaheuristics are more flexible and robust than simple heuristics, they often require significant computational resources and domain expertise to design effective operators and parameters.

Hybrid Methods Despite their differences, all three approaches mentioned so far face inherent limitations: exact methods struggle with scalability, heuristics lack guarantees, and metaheuristics demand extensive tuning. These challenges have motivated the development of hybrid methods that combine the three approaches [BR16]. For example, exact methods can be used to explore subspaces of the solution space (e.g., MIP-Based Large Neighborhood Search [PR10]) and heuristics can be used to improve the solutions found during the branch-and-bound search of an exact ILP solver (e.g., Feasibility Jump [LS23]). A recent trend in the hybridization of CO methods is the integration of machine learning techniques into classical optimization methods, which will be explored in detail in the next section.

1.1.4 Machine Learning in Combinatorial Optimization

Machine Learning (ML) techniques are increasingly used to tackle CO problems [BLP21]. Learning methods promise to improve the efficiency of classical algorithms by exploiting learned problem structures and patterns. The learning process can be supervised, where the model learns from expert demonstrations, or guided by rewards in a reinforcement learning framework.

Common Approaches

There are numerous approaches to integrating ML into methods for CO. It can be utilized to automatically tune the parameters of CO algorithms, such as selecting the best branching strategy in branch-and-bound or determining when to apply decomposition techniques. Additionally, ML can learn policies for decision-making within CO algorithms, such as selecting which variable to branch on or which heuristic to apply at a given point in the search process. These policies can be acquired through imitation learning, where the model mimics expert decisions, or through reinforcement learning, which involves learning from trial and error. In some instances, ML models are trained to directly output solutions to CO problems, which we call end-to-end learning.

Graph Neural Networks in Combinatorial Optimization

A Graph Neural Network (GNN) [WCPZ22] is a class of neural networks designed to process graph-structured data by iteratively updating node features through message passing—a mechanism where nodes aggregate information from their neighbors to refine their representations. This structure-aware approach enables GNNs to capture relational patterns and dependencies within graphs, making them versatile for tasks like node classification, edge prediction, and graph-level inference.

Some CO problems are naturally formulated as graph problems, and many others can be translated to a graph problem. An example of such translation (formally called a reduction) is the TSP being equivalent to finding a Hamiltonian cycle of minimum cost on a directed graph. But other reductions are more complex and artificial, like for Packing Problems [LMMV10]. Because of this graph reformulation of CO problems, Graph Neural Networks (GNNs) are a popular architecture for ML models in CO algorithms.

Generative Models for Combinatorial Optimization - Difusco

Generative models have emerged as a powerful strategy for addressing combinatorial optimization (CO) problems, with denoising diffusion models recognized as the most notable among these approaches. This thesis will concentrate solely on denoising diffusion models, which will be referred to simply as diffusion models since there is no confusion possible.

Denoising diffusion models are generative models that learn to reconstruct data by reversing a gradual noising process, effectively generating new samples from random noise. They can be applied to CO problems by learning to sample solutions directly from a distribution of high-quality solutions. In this setting, diffusion models are usually equipped with a GNN to learn the denoising process. This is the approach taken by Difusco [SY23], the main reference of this thesis, and we will build upon it. Another possibility is to use an unsupervised diffusion approach, where the diffusion process is guided by a problem-specific loss [SHL24].

Compared to other end-to-end learning approaches, diffusion models are able to generate a solution in a single denoising pass, as opposed to the quadratic sequential autoregressive generation of other models. Moreover, they achieve this without the assumption of independence between the solution components, which is a common simplification in other models. As a diffusion model, Difusco claims to be able to generate a multimodal distribution of high-quality solutions, which can be later refined through local search procedures.

Difusco represented a major step forward in neural solvers, achieving state-of-the-art results on the Traveling Salesperson Problem (TSP). It inspired a series of follow-up works, including improvements to the diffusion process [LGWY23], tricks for out-of-distribution generalization [YZH+24], and the hybridization of classical algorithms such as MIP solvers with diffusion-based components [FSY24].

Limitations of Machine Learning in Combinatorial Optimization

While ML offers promising avenues for solving CO problems, several critical limitations hinder its widespread adoption and reliability in this domain. For instance, at this time, there is no state-of-the-art solver for CO that mostly or only relies on ML components. These limitations are summarized in the following list:

1. **Generalization and Scalability**: Machine Learning models in CO often face significant challenges in generalizing beyond the specific instance sizes or distributions encountered during training. For instance, models trained on small-scale TSPs frequently exhibit diminished performance when applied to larger instances. Moreover, even when dealing with instances of the same size, variations in problem structures—such as differences between sparse and dense graphs—can lead to performance degradation. Likewise, applying an ML model trained on TSPs to a different problem, like the Vehicle Routing Problem, will likely result in suboptimal outcomes due to the lack of transferability across distinct problem types. This challenge is further compounded by the absence of theoretical guarantees regarding generalization bounds or computational complexity for learned heuristics. This limitation aligns with the "No Free Lunch" theorem, which asserts that no single optimization algorithm can consistently outperform others across all possible problems without specific prior knowledge.

2. **Feasibility and Constraint Satisfaction**: Unlike traditional CO algorithms, ML-based heuristics do not inherently guarantee the feasibility of the generated solutions. In the design of neural networks to be trained via gradient descent, the choice of activation functions and loss functions can be prone to prioritize differentiable operations, often at the expense of enforcing discrete constraints. This can result in solutions that violate critical problem requirements, requiring post-hoc repair of solutions via heuristics.

3. **Model Architecture Suitability**: ML architectures successful in other domains (e.g., CNNs for images or transformers for sequence data) may lack the inductive biases needed for CO. While graph neural networks (GNNs) show promise for CO problems in graph instances, designing architectures that capture combinatorial-specific symmetries, sparsity, or variable dependencies remains non-trivial. Furthermore, training algorithms (e.g., SGD) may require adaptation to effectively navigate discrete, non-convex CO solution spaces.
4. **Integration with Classical Algorithms**: Pure end-to-end ML approaches often lack the robustness of classical CO algorithms. Hybrid strategies—combining learned policies with exact methods—are essential to retain theoretical guarantees. However, seamless integration requires overcoming compatibility issues, such as embedding differentiable components within discrete optimization frameworks.
5. **Theoretical and Practical Uncertainty**: The absence of rigorous performance bounds for ML-based CO methods introduces uncertainty in real-world deployment. While ML can accelerate or improve heuristic decisions, its "black-box" nature complicates verification.

1.2 Motivation and Research Gap

In this thesis, we investigate the application of Denoising Diffusion Models to CO problems, with a particular focus on advancing the Difusco framework [SY23].

1.2.1 Strengths and Limitations of Difusco

As highlighted before, Difusco represents a significant advancement in neural CO solvers, offering two key advantages over existing approaches:

- **The multimodal property**: By leveraging the stochastic denoising process of diffusion models, the framework generates distinct high-quality solutions when initialized with different noise samples. This property enables the model to capture a diverse, multimodal distribution of near-optimal solutions, addressing a critical limitation of deterministic or unimodal neural solvers.
- **Computational efficiency**: Unlike autoregressive methods that suffer from quadratic time complexity in the sequential generation of solution components, Difusco provides a complete solution in a single denoising pass. Notably, it achieves this without sacrificing the ability to model dependencies between solution components, which is a common trade-off in non-autoregressive approaches.
- **Scalability**: GPU-based implementations can efficiently process batches of multiple instances in parallel, typically exhibiting sublinear scaling of runtime with respect to batch size. Consequently, the diffusion component of a larger algorithm that incorporates Difusco can be effectively parallelized across multiple instances.

Despite these strengths, Difusco and other ML-based neural solvers also face two main limitations that highlight the need for further refinement:

- **Over-reliance on subsequent heuristic refinement**: Generated solutions are not guaranteed to satisfy problem-specific constraints, requiring post-hoc feasibility repair mechanisms. To address this, the framework integrates two-stage refinement: feasibility heuristics correct constraint violations, while subsequent local search procedures (e.g., 2-opt or Monte Carlo Tree Search) iteratively optimize solution quality. Notably, Difusco's state-of-the-art performance on the Traveling Salesperson Problem (TSP) is achieved only when paired with these subsequent refinement steps. This underscores a critical trade-off: while diffusion models excel at exploring diverse solution spaces, their practical effectiveness hinges on hybridization with classical optimization techniques.

- **Problem-specific heuristics**: Both the feasibility repair mechanisms and local search procedures are tailored to the specific structure and constraints of the target problem. This limits the framework's generalizability, as adapting Difusco to new CO problems requires designing and integrating appropriate heuristics for each new domain.

1.2.2 Bridging ML and Metaheuristics: A General Framework

We do not address the issue of over-reliance on subsequent heuristic refinement, as this is a challenge common to most ML-based CO solvers, not specific to Difusco. This reliance stems from the broader research community's emphasis on achieving performance comparable to state-of-the-art classical solvers, which currently outperform purely ML-based approaches [WWW^{+}24]. Addressing this limitation remains an open research question, requiring advancements in both ML architectures and training paradigms.

Instead, we focus on the second limitation: the lack of a *unified framework* that integrates the generative power of diffusion models with the problem-independent exploratory strengths of classical metaheuristics. This limitation is particularly critical as it restricts the generalizability and adaptability of existing approaches across diverse CO problems.

To address this gap, this thesis proposes a *Diffusion-Based Evolutionary Algorithm (DEA)* framework. Unlike existing methods, DEA embeds diffusion models directly into the evolutionary search process, leveraging their generative capabilities to guide key operators such as initialization, crossover, and mutation. This approach aims to:

- Be adaptable to multiple CO problems with minimal architectural changes. Once trained on a specific problem, the diffusion model serves as a black-box operator, while the EA operates as a general-purpose metaheuristic, requiring no significant modifications across problem domains.

- Leverage the multimodal sampling capability of diffusion models to explore varied regions of the solution space by keeping a solution pool of diverse solutions. The EA then refines these solutions through its operators.

- Benefit from hardware accelerators by keeping the population in the GPU memory.

1.3 Aim of the Thesis

1.3.1 Objectives and Research Questions

The primary objective of this thesis is to bridge the gap between generative machine learning models and classical metaheuristics by developing a novel framework that integrates diffusion models with EAs. This integration aims to address the critical limitations of existing ML-based CO solvers: their reliance on problem-specific local search heuristics. To systematically evaluate this integration, the research addresses the following objectives and questions:

1. **Understanding Difusco's strengths and limitations**: Study the limitations of Difusco by benchmarking it on different CO problems and performing an ablation study that isolates the contribution of the diffusion components.

    - How much of the success of Difusco is due to the posterior local search heuristics?
    - Is the multimodal property of Difusco observable?
    - How do Difusco's runtimes scale with the problem size and the number of processed instances in parallel?

2. **Evaluating the DEA Framework**: Implement the DEA and benchmark it on different CO problems against classical methods, pure ML-based solvers, and hybrid baselines.

    - How does the DEA compare to other solvers in terms of solution quality and computational time?
    - How does the equivalent method without diffusion-based operators perform? What is the contribution of the diffusion components?
    - How does the DEA perform in terms of generalization to out-of-distribution instances?

1.3.2 Contributions and Expected Outcomes

This thesis makes the following primary contributions to the field of ML-based CO:

- **A Novel Diffusion-Based Evolutionary Algorithm (DEA) Framework**: The central contribution is the development of a general and adaptable framework that seamlessly integrates diffusion models into the core operators of an EA. This DEA framework leverages the multimodal generative capabilities of diffusion models for solution initialization, crossover, and mutation within the EA search process. The framework is designed to be problem-agnostic, meaning that it does not rely on problem-specific heuristics to achieve good performance.

- **Enhanced Solution Quality and Exploration**: We expect the DEA framework to achieve improved solution quality compared to plain Difusco and classical EAs with naive operators. By embedding diffusion models, the EA is expected to (1) start with a diverse set of high-quality solutions coming from diffusion-sampling, and (2) benefit from a more informed exploration of the solution space thanks to diffusion-based operators, allowing the EA to refine the initial population and converge to a candidate solution. However, we do not expect the EA-DDM framework to outperform highly specialized state-of-the-art solvers for specific problems. Instead, we aim to achieve competitive performance across a range of CO problems while offering a more general and scalable approach compared to current end-to-end ML-based methods.

- **Improved Generalization Capacity**: We anticipate that the DEA will demonstrate enhanced generalization capabilities compared to purely end-to-end ML-based solvers. The inherent exploratory nature of the EA, combined with the generative power of diffusion models, should enable the framework to adapt more effectively to out-of-distribution instances and potentially different CO problem types, offering a more robust and adaptable solution approach.

- **Scalability and Hardware Acceleration**: The DEA framework is designed to leverage the parallel processing capabilities of GPUs. By maintaining the solution population and performing the diffusion inference on GPU, we expect the framework to scale efficiently to larger problem instances, a critical requirement for real-world CO applications.

- **Bridging the Gap between ML-based and Classical Optimization**: This work aims to contribute to bridging the gap between ML methods and classical optimization techniques. By demonstrating the effective hybridization of diffusion models with EAs, this thesis explores a promising direction for developing more general, robust, and performant CO solvers that combine the strengths of both paradigms.

1.4 Thesis Structure

The remainder of this thesis is organized as follows:

Chapter 2 presents a comprehensive literature review on CO, and on machine learning approaches for CO, with special focus on diffusion models. The chapter examines existing hybrid approaches and identifies the research gaps that motivate our work.

Chapter 3 details the proposed Diffusion-Based Evolutionary Algorithm (DEA) framework. It describes the integration of diffusion models into evolutionary operators, the architecture design choices, and training procedures.

Chapter 4 presents the experimental evaluation of the DEA framework. The chapter reports the results of comparative studies and ablation analyses, addressing the research questions outlined in Section 1.3.1.

Chapter 5 summarizes the key findings and contributions of this thesis. It discusses the implications of the results, acknowledges limitations, and suggests directions for future research in the integration of diffusion models with classical optimization techniques.

The Appendix provides supplementary material, including the hyperparameters used for the experiments, and background on Graph Neural Networks.

CHAPTER 2

Related Work

This chapter reviews the relevant literature for this thesis. In Section 2.1, we establish the necessary background on combinatorial optimization, including formal problem definitions and classical solution methods. Section 2.2 examines and categorizes machine learning approaches for combinatorial optimization. Section 2.3 introduces the fundamentals of diffusion models, while Section 2.4 dives into their application to combinatorial optimization, with particular emphasis on the Difusco framework and subsequent developments. Finally, Section 2.5 explores evolutionary algorithms, their components, and their application to optimization problems like the Maximum Independent Set.

2.1 Combinatorial Optimization

In this section, we begin by establishing the necessary notation and providing the formal background on combinatorial optimization. Next, we outline the two primary CO problems that are the focus of this thesis. Finally, we conduct a review of classical methods for CO, encompassing exact methods, heuristics, and metaheuristics.

2.1.1 Preliminaries and Problem Definitions

We first introduce notation and give the necessary formal background on CO, the different machine learning regimes, and GNNs.

Let $[n] = \{1, \dots, n\} \subset \mathbb{N}$ for $n \geq 1$. For a (finite) set S, we denote its power set as 2^S. A graph G is a pair (V, E) with a finite set of nodes V and a set of edges $E \subseteq V \times V$. We denote the set of nodes and the set of edges of G by $V(G)$ and $E(G)$, respectively. A labeled graph G is a triplet (V, E, l) with a label function $l : V(G) \cup E(G) \to \Sigma$, where Σ is some finite alphabet. Then, $l(x)$ is a label of x, for $x \in V(G) \cup E(G)$. Note that x here can be either a node or an edge. The neighborhood of v in $V(G)$ is denoted by $N(v) = \{u \in V(G) \mid (v, u) \in E(G)\}$.

Definition 2.1.1 (Combinatorial optimization instance) *An instance of a CO problem is a tuple (Ω, F, w), where: Ω is a finite set, $F \subseteq 2^\Omega$ is the set of feasible solutions, and $c : 2^\Omega \to \mathbb{R}$ is a cost function defined by $c(S) = \sum_{\omega \in S} w(\omega)$ for $S \in F$.*

We continue with the definition of the two examples that we will use throughout this thesis.

Example 2.1.1 (Traveling Salesperson Problem (TSP)) *An instance of the TSP is defined as a complete directed graph $G = (V, E)$ where $E(G) = \{(u, v) \mid u, v \in V(G)\}$ and a cost function $w : E(G) \to \mathbb{R}$. The objective is to find a permutation $\sigma : \{0, \ldots, n-1\} \to V$ that minimizes the total travel cost given by:*

$$\mathit{Cost}(\sigma) = \sum_{i=0}^{n-1} w(\sigma(i), \sigma((i+1) \mod n))$$

where $n = |V|$ is the number of nodes in the graph.

It can be shown that the Hamiltonian cycle problem is NP-complete, which implies the NP-hardness of TSP [Kar72]. In complexity theory, one refers to the optimization version of such difficult problems as NP-hard, while their decision version is generally NP-complete.

Example 2.1.2 (Maximum Independent Set (MIS)) *An instance of the Maximum (Weighted) Independent Set (MWIS or MIS) problem consists of a graph $G = (V, E)$ and a weight function $w : V \to \mathbb{R}$. The goal is to find an independent set $S \subseteq V$ (i.e., no two vertices in S are adjacent) that maximizes the total weight:*

$$\mathit{Cost}(S) = \sum_{v \in S} w(v)$$

Formally, S is an independent set if $(u, v) \notin E$ for all $u, v \in S$. In the unweighted case, we have $w(v) = 1$ for all $v \in V$.

It can be shown that the problem of finding a maximum independent set is NP-hard [Kar72].

2.1.2 A Review of Classical Methods for Combinatorial Optimization

We have already mentioned the three main classes of classical methods for CO: exact methods, heuristics, and metaheuristics.

Exact Methods

Exact methods aim to find provably optimal solutions for CO problems, and Integer Linear Programming (ILP) or Constraint Programming (CP) are two popular approaches. In this section, we give a brief overview of Integer Linear Programming because it is the most relevant to our work.

Integer Linear Programming A linear program aims at optimizing a linear cost function over a feasible set described as the intersection of finitely many half-spaces, i.e., a polyhedron. Formally, we define an instance of a linear program as follows:

Definition 2.1.2 (Linear Programming Instance) *An instance of a linear program (LP) is a tuple (A, b, c), where A is a matrix in $\mathbb{R}^{m \times n}$, and b and c are vectors in $\mathbb{R}^m$ and $\mathbb{R}^n$, respectively.*

The associated optimization problem asks to minimize a linear objective over a polyhedron. That is, we aim to find a vector $x \in \mathbb{R}^n$ that minimizes $c^T x$ over the feasible set

$$X = \{x \in \mathbb{R}^n \mid A_j x \leq b_j \text{ for } j \in [m] \text{ and } x_i \geq 0 \text{ for } i \in [n]\}$$

In practice, LPs are solved using the Simplex method or polynomial-time interior-point methods [BT97]. Due to their continuous nature, LPs cannot encode the feasible set of a CO problem. Hence, we extend LPs by adding *integrality constraints*, i.e., requiring that the value assigned to each variable is an integer. Consequently, we aim to find the vector $x \in \mathbb{Z}^n$ that minimizes $c^T x$ over the feasible set

$$X = \{x \in \mathbb{Z}^n \mid A_j x \leq b_j \text{ for } j \in [m], x_i \geq 0 \text{ and } x_i \in \mathbb{Z} \text{ for } i \in [n]\}.$$

Such integer linear optimization problems are solved by tree search algorithms, e.g., branch-and-bound algorithms [LD60]. By dropping the integrality constraints, we again obtain an instance of an LP, which we call relaxation. Solving the LP relaxation of an ILP provides a valid lower bound on the optimal solution of the problem, i.e., an optimistic approximation, and the quality of such an approximation is largely responsible for the effectiveness of the search scheme.

Example 2.1.3 (An ILP formulation for the TSP) *We provide an ILP formulation for the asymmetric TSP. Let x_{ij} be a binary variable that equals 1 if the cycle goes from city i to city j, and 0 otherwise. Let $w_{ij} > 0$ be the cost or distance of traveling* *from city i to city j, $i \neq j$. Then, the TSP can be written as the following ILP:*

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} x_{ij} & \\
s.t. \quad & \sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1 & \forall j \in [n], \\
& \sum_{j=1, j \neq i}^{n} x_{ij} = 1 & \forall i \in [n], \\
& \sum_{i \in Q} \sum_{j \notin Q} x_{ij} \geq 1 & \forall Q \subsetneq [n], |Q| \geq 2, \\
& x_{ij} \in \{0,1\} & \forall i,j \in [n], i \neq j.
\end{aligned}$$

Example 2.1.4 (An ILP formulation for the MIS) *Consider a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. We define a binary variable x_i for each vertex $i \in V$ such that $x_i = 1$ if vertex i is included in the independent set, and 0 otherwise. The goal is to maximize the size of the independent set, which can be formulated as the following ILP:*

$$\begin{aligned}
\max \quad & \sum_{i \in V} x_i & \\
s.t. \quad & x_i + x_j \leq 1 & \forall (i,j) \in E, \\
& x_i \in \{0,1\} & \forall i \in V.
\end{aligned}$$

Solving ILPs is NP-hard. There exist several ILP solvers, such as CPLEX, Gurobi, and SCIP, which are based on branch-and-bound algorithms. For most problems, solving to optimality quickly becomes intractable as the instance size increases. Reformulation techniques can be used to transform ILPs into equivalent problems with stronger relaxations (e.g., [SSHR24]), or hyperparameter tuning can be used to tune the solver for the specific problem at hand (e.g., [IL20]). In practice, solvers are then commonly used with a time limit, and the best solution found is returned, which is often of acceptable quality.

Constraint Programming Another general-purpose approach for solving CO problems is Constraint Programming (CP). CP is a declarative programming paradigm that allows us to express combinatorial problems by expressing the constraints that must be satisfied by the solution. A natural extension of CSPs are constrained optimization problems, which incorporate an objective function. The goal in this case becomes finding a feasible assignment that minimizes (or maximizes) the objective function while satisfying all constraints. Solving CSPs is also NP-hard. CP-SAT by OR-Tools or Gecode are two popular solvers for this class of problems.

Algorithm 2.1: Constructive Heuristic

```
   Output: A feasible solution
1  Initialize partial solution $s_p \leftarrow \emptyset$;
2  while $Ext(s_p) \neq \emptyset$ do
3      Select component $c \leftarrow$ Select(Ext($s_p$));
4      Add $c$ to $s_p$;
5  end
6  return $s_p$;
```

Heuristics

Heuristics are practical methods designed to find high-quality solutions to CO problems efficiently, particularly when exact methods become computationally intractable. These methods prioritize speed and practicality over theoretical guarantees of optimality. Below, we formalize key concepts and frameworks that underpin heuristic approaches.

Constructive Heuristics Constructive heuristics generate solutions incrementally from an initially empty partial solution. Given a problem instance I for a CO problem P, let C denote the *complete set of solution components* for I. A valid solution s is represented as a subset $s \subseteq C$. For example, in the Traveling Salesperson Problem (TSP), C corresponds to the set of all edges in the input graph G, and a solution s is a subset of edges forming a valid tour.

The algorithmic framework for constructive heuristics is outlined in Algorithm 2.1. Starting with an empty partial solution s_p, the heuristic iteratively extends s_p by selecting a component c from the set of feasible extensions $\mathrm{Ext}(s_p) \subseteq C$. The process terminates when no further extensions are possible. The selection mechanism often employs a *greedy function* to prioritize components that locally optimize an objective metric.

Constructive heuristics accept randomization in the selection of the component to extend the partial solution, which is a common practice to avoid getting stuck in local optima. Constructive heuristics are often used as a building block for more complex heuristics or metaheuristics.

Example 2.1.5 (Nearest Neighbor Heuristic for TSP) *The Nearest Neighbor heuristic for TSP starts with an empty partial solution and iteratively selects the nearest unvisited node to the current partial solution until all nodes are included in the tour.*

Example 2.1.6 (Greedy Heuristic for MIS) *A greedy heuristic for MIS starts with an empty partial solution and iteratively selects the still available vertex with the lowest degree in the remaining graph until no more vertices can be added to the independent set. A vertex is marked as unavailable if it is adjacent to any vertex in the current partial solution.*

Algorithm 2.2: Local Search

```
Input: Initial solution $s$, neighborhood function $N$
Output: A local optimum solution
1 while $\exists\ s' \in N(s)$ such that $f(s') < f(s)$ do
2     $s \leftarrow$ ChooseImprovingNeighbor($N(s)$);
3 end
4 return $s$;
```

Local Search Local search (LS) methods iteratively improve an initial solution by exploring its *neighborhood*, defined via a neighborhood function:

Definition 2.1.3 (Neighborhood Function) *A neighborhood function $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$ maps a solution $s \in \mathcal{S}$ to a set of neighboring solutions $\mathcal{N}(s) \subseteq \mathcal{S}$. Neighbors are generated by applying a move operator (e.g., swapping nodes in a TSP tour).*

LS terminates when no better solution exists in the neighborhood of the current solution (i.e., a *local minimum* is reached):

Definition 2.1.4 (Local Minimum) *A solution $\hat{s}$ is a local minimum w.r.t. $\mathcal{N}$ if $f(\hat{s}) \leq f(s)\,\forall s \in \mathcal{N}(\hat{s})$. If $f(\hat{s}) < f(s)$, $\hat{s}$ is a strict local minimum.*

Example: 2-opt for TSP The 2-opt heuristic removes two edges from the current tour and reconnects the paths to form a shorter tour [AL03].

Local search procedures are a generic tool to improve the quality of a given solution and can be applied in a wide variety of settings. They are often used in combination with constructive heuristics to further improve the quality of initial solutions. LS is commonly used in conjunction with ML-based approaches for CO, where it is used to refine the solution generated by the ML model [SY23].

There exist algorithms that systematically exploit local search by exploring multiple neighborhoods to enhance solution quality. A notable example is the Variable Neighborhood Descent (VND) algorithm, which systematically explores multiple neighborhood structures $\{\mathcal{N}_1, \ldots, \mathcal{N}_{k_{\max}}\}$, often ordered by increasing complexity.

Metaheuristics

Metaheuristics are high-level algorithmic frameworks designed to guide and enhance the search for high-quality solutions to CO problems. Unlike problem-specific heuristics, metaheuristics are broadly applicable across domains, offering strategies to balance exploration (diversifying the search across the solution space) and exploitation (intensifying the search around promising regions). Emerging in the 1970s, these methods address the limitations of basic heuristics—such as stagnation in local optima—by incorporating mechanisms like randomization, memory, and adaptive learning. Metaheuristics are particularly effective for NP-hard problems where exact methods are impractical due to scalability constraints.

Below are brief descriptions of widely used metaheuristics, emphasizing their core principles:

- **Evolutionary Algorithms (EA)** [BFM97]: EAs evolve a population of solutions through selection, crossover, and mutation, mimicking natural evolution. For example, in TSP, solutions (tours) are recombined via edge crossover to generate offspring. We will discuss EAs in more detail in Section 2.5.
- **Simulated Annealing (SA)** [KGV83]: Inspired by metallurgical annealing, SA probabilistically accepts worse solutions during the search to escape local optima, controlled by a temperature parameter that decreases over time.
- **Ant Colony Optimization (ACO)** [DS04]: ACO mimics ant foraging behavior by depositing "pheromones" on promising solution components (e.g., edges in TSP), reinforcing high-quality paths over iterations.
- **Tabu Search (TS)** [Glo89]: TS uses short-term memory to avoid revisiting recently explored solutions, enabling systematic exploration of the search space.

These metaheuristics provide flexible frameworks for tackling CO problems, often serving as foundational components in hybrid algorithms.

2.2 Machine Learning Approaches in Combinatorial Optimization

This section provides a general overview of ML techniques applied to CO, classifying approaches based on learning methodologies and algorithmic integration. We use the classification proposed in [BLP21] and extend it to include recent advances in the field.

2.2.1 Type of Learning

The first categorization is based on the type of learning employed: imitation learning, reinforcement learning, and gradient-based learning.

Imitation learning Imitation learning, also known as learning from demonstration, leverages expert knowledge on the CO task to train an ML model. The "expert" can be a traditional, well-performing CO algorithm or heuristic. The ML model's goal is to mimic the expert's decisions, effectively learning a mapping from the problem state to the expert's chosen action. This is typically framed as a supervised learning problem, where the expert's decisions provide the target labels for training.

The primary advantage of imitation learning is its relative simplicity and data efficiency, especially when a good expert is available. However, the performance of the learned model is fundamentally bounded by the performance of the expert; it generally cannot surpass the expert's capabilities and struggles to generalize to unseen problem instances. A common application is to approximate computationally expensive components of existing algorithms, such as strong branching or cut selection in Mixed-Integer Programming (MIP) [KBS$^+$16, GCF$^+$19, HWL$^+$21]. Another example is to approximate the Semidefinite Programming (SDP) relaxation, which is generally computationally expensive [BLMBT18].

Reinforcement learning Reinforcement Learning (RL) offers a contrasting approach where an agent learns to make decisions through interaction with an environment. In the context of CO, the environment typically represents the state of the optimization process (e.g., the current state of a branch-and-bound tree, a partially constructed solution to a problem instance). The agent's actions correspond to decisions within the CO algorithm (e.g., selecting a branching variable, choosing the next node to visit in a TSP). The agent learns a policy that maximizes a cumulative reward signal, which reflects the quality of the solution or the efficiency of the search process.

Unlike imitation learning, RL does not require an expert demonstration. The agent learns through trial and error, potentially discovering novel strategies that outperform traditional heuristics. However, RL often requires significantly more training data (experience) and can be sensitive to the design of the reward function and exploration strategy. RL has been successfully applied to problems like the TSP [KvHW19, BPL$^+$17, ZHZ$^+$21], vehicle routing [KvHW19], and job shop scheduling [ZSC$^+$20]. The use of graph neural networks (GNNs) has become increasingly popular in this area, allowing the RL agent to effectively process the graph-structured representations common in CO problems [KDZ$^+$17].

Gradient-based learning Gradient-based learning for CO is a type of unsupervised learning that involves parameterizing the solution space using neural networks and optimizing these parameters to minimize the objective function of the CO problem directly. The main challenge of this approach is handling the discrete nature of the CO problem in order to apply gradient-based optimization methods. Relaxing a CO problem into a continuous space (e.g., via penalty terms for constraints, relaxing integrality constraints) is a common approach.

Examples of early work include [KL20], [WWY$^+$22], and [SBK22], which study relaxations and optimization methods in order to apply an unsupervised learning approach to CO problems. The combination of unsupervised learning and diffusion models has been explored in [SHL24]. Recent works have tried to solve general Mixed-Integer Programs (MIPs) using unsupervised learning [GWL$^+$25].

2.2.2 Algorithmic Integration

The second categorization is based on the nature of the algorithmic integration between the ML model and the CO algorithmic framework: end-to-end learning, learning to configure algorithms, and ML alongside optimization algorithms.

End-to-end learning In this paradigm, the ML model is trained to directly map a problem instance, represented by its defining parameters, to a solution. The ML model effectively constitutes the solver, circumventing traditional CO techniques. This approach is often adopted when computational speed is paramount, and relaxations of optimality or feasibility guarantees are acceptable. Early work in this area has explored architectures such as pointer networks for sequence-to-sequence mapping in problems like the TSP [VFJ15]. More recently, architectures such as Difusco [SY23] or the unsupervised learning examples mentioned above have gained popularity and are good examples of this paradigm.

Learning to configure algorithms This approach leverages ML to inform the configuration and parameterization of existing CO algorithms. The ML model does not directly solve the problem but rather provides insights that guide the CO solver's execution. This is closely related to the field of automated algorithm configuration [Hoo12, BKK$^+$16]. The learned model predicts optimal or near-optimal parameter settings, algorithmic choices, or strategic decisions based on features extracted from the problem instance. Examples include predicting the efficacy of Dantzig-Wolfe decomposition [KLP17], or designing a schedule of heuristics for mixed-integer quadratic programs [CKG$^+$21]. There is also work on using ML to configure MIP solvers for large-scale instances [KLKea24], which falls in the category of classical ML-based hyperparameter optimization.

With the recent advances in the field of Large Language Models, there is growing interest in applying them to solve CO problems. LLMs, pretrained on vast corpora of text and code, have shown surprising potential in CO tasks due to their ability to reason about symbolic representations of problems. An example is their application as hyper-heuristics, where LLMs can act as heuristic algorithm designers by proposing problem-specific components [YWC$^+$24]. Another example is their application as cut selection heuristics in MIPs, where LLMs can predict which cutting planes (e.g., Gomory cuts) to apply by reasoning about the problem structure [LLW$^+$24].

ML alongside optimization algorithms This category encompasses approaches where the ML model and the CO algorithm engage in a repeated, iterative interaction. The CO algorithm frequently queries the ML model during its execution, typically to make sequential decisions that influence the algorithm's trajectory. The ML model receives a representation of the algorithm's current state as input and outputs a decision or a set of scores guiding the next action. This is particularly relevant in tree search algorithms like branch-and-bound, where ML can be used for variable branching, node selection, or heuristic control. [SALYS24] is a comprehensive survey of ML-augmented branch-and-bound algorithms.

2.3 Diffusion Models

Diffusion models have emerged as a powerful framework for generative modeling, achieving state-of-the-art results in image synthesis [HJA20], audio generation [CZZ+21], and molecular design [HSVW22]. Unlike autoregressive models or transformers, they do not inherently rely on self-attention mechanisms, making them computationally flexible for diverse data modalities [SSDK+21]. Their iterative denoising process enables high-quality generation across applications: creating photorealistic images from text prompts (DALL · E 2 [RDN+22], Stable Diffusion [RBL+22]), synthesizing music waveforms (AudioLM [BMV+22]), and generating 3D molecular structures by diffusing atomic coordinates [HSVW22]. A key advantage lies in their stable training dynamics compared to GANs, avoiding mode collapse through a structured noise-to-data transition [SDWMG15].

The core principle of denoising diffusion models is simple: progressively add noise to data until it becomes pure noise (forward process), then learn to reverse this process (backward process) to generate new data from noise. This two-step approach can be understood through the lens of image denoising—where a clean image is gradually corrupted with noise, and a model learns to recover the original by iteratively removing the noise. Figure 2.1 illustrates this process using a cat image, showing how the original image is systematically corrupted and then recovered through denoising.

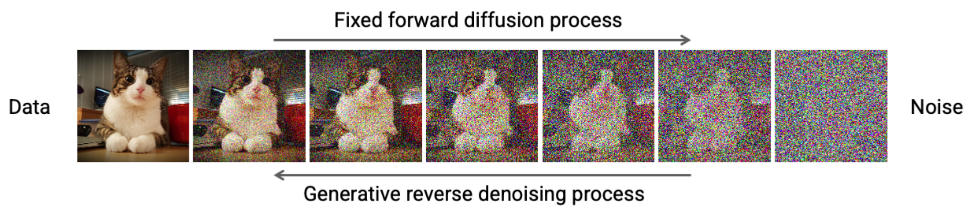
Fixed forward diffusion process
Data
Noise
Generative reverse denoising process

Figure 2.1: Transformation of a cat image through the noising and denoising process.

Fundamentals of denoising diffusion models At the core of diffusion models are two Markov processes, represented by the distributions q and p_θ:

1. **Forward Diffusion Process (q):**
   This is a fixed, known process that gradually corrupts the original data x_0 by adding Gaussian noise over T steps. At each step, the conditional probability is defined as
$$q(x_t \mid x_{t-1}) = \mathcal{N}\Big(x_t; \sqrt{1-\beta_t}\, x_{t-1},\ \beta_t \mathbf{I}\Big),$$

where $\beta_t \in (0,1)$ determines the noise schedule and $\mathbf{I}$ denotes the identity matrix. By cumulatively applying this process, the distribution of x_t conditioned on x_0 is given by

$$q(x_t \mid x_0) = \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t}\, x_0,\, (1-\bar{\alpha}_t)\,\mathbf{I}\right),$$

with $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$ representing the cumulative product of the α_s values. Both $q(x_t \mid x_{t-1})$ and $q(x_t \mid x_0)$ are known and fully specified.

The choice of noise schedule $\{\beta_t\}_{t=1}^{T}$ significantly impacts model performance. Linear schedules (where β_t increases linearly with t) were initially proposed [HJA20], but cosine schedules [ND21] often yield better results by allocating more steps to critical noise levels. Some recent approaches [KSPH21] even learn the noise schedule as part of the optimization process, adapting it to the specific data distribution.

2. **Reverse Process ($p \approx p_\theta$):**
In contrast, the reverse process is learned and aims to invert the forward diffusion. It is parameterized by a neural network with parameters θ and defined as

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t),\, \Sigma_\theta(x_t, t)\right).$$

Here, $p_\theta(x_{t-1} \mid x_t)$ represents the conditional distribution that the network approximates to progressively reconstruct x_{t-1} from x_t. The neural network conditions on both the noisy input x_t and the time step t, typically through sinusoidal time embeddings [VSP+17] that encode t as a high-dimensional vector.

Unlike the fixed forward process q, the reverse process is not known a priori and is learned by optimizing a variational objective that encourages p_θ to closely invert the noising procedure defined by q. While some diffusion models use a fixed variance $\Sigma_\theta(x_t, t) = \sigma_t^2 \mathbf{I}$ derived from the forward process, others learn it jointly with the mean $\mu_\theta(x_t, t)$.

The architecture choice for the neural network θ depends on the data modality. For images, U-Net architectures [RFB15] are widely adopted due to their ability to capture multiscale features through skip connections, preserving both local details and global context. For CO problems, Graph Neural Networks (GNNs) [SY23] are preferred as they naturally enable the model to reason about node interactions and graph topology.

Training objective Diffusion models are typically trained by maximizing a variational lower bound—commonly referred to as the Evidence Lower Bound (ELBO)—on the log-likelihood of the data:

$$\mathbb{E}_{q(x_0)}\left[\log p_\theta(x_0)\right] \geq \tag{2.1}$$

$$\mathbb{E}_q\left[\underbrace{\log p_\theta(x_T)}_{\text{Prior Matching}} + \sum_{t>1} \underbrace{D_{KL}\Big(q(x_{t-1} \mid x_t, x_0) \,\|\, p_\theta(x_{t-1} \mid x_t)\Big)}_{\text{Denoising Matching}} + \underbrace{\log p_\theta(x_0 \mid x_1)}_{\text{Reconstruction}}\right] \tag{2.2}$$

The ELBO serves as a tractable surrogate for the intractable marginal log-likelihood $\log p_\theta(x_0)$. A proof of the inequality can be found in [Cha24]. The ELBO decomposes into three primary components:

1. **Prior Matching Term ($\log p_\theta(x_T)$):** This term ensures that the distribution of the final latent variable x_T (obtained after applying the forward diffusion process) conforms to a predefined prior—typically a standard Gaussian $\mathcal{N}(0, \mathbf{I})$. This alignment is essential for initializing the reverse process in a consistent latent space. In practice, for sufficiently large T, x_T approaches a standard Gaussian regardless of x_0, making this term negligible.
2. **Denoising Matching Term ($D_{KL}(q(x_{t-1} \mid x_t, x_0) \,\|\, p_\theta(x_{t-1} \mid x_t))$):** For each time step t, this term quantifies the discrepancy (via the Kullback-Leibler divergence) between the true posterior $q(x_{t-1} \mid x_t, x_0)$, derived from the forward process, and the learned reverse process $p_\theta(x_{t-1} \mid x_t)$. Minimizing this divergence enforces that the reverse process effectively inverts the noise addition. This term constitutes the core of the training objective, as it directly guides the model to learn the denoising dynamics.
3. **Reconstruction Term ($\log p_\theta(x_0 \mid x_1)$):** This term encourages the model to accurately reconstruct the original data x_0 from the slightly noised data x_1. It acts as a final refinement step, ensuring that the reverse process, when applied iteratively, recovers the original data with high fidelity. In many implementations, this term is implicitly optimized through the denoising matching term, especially when using the simplified noise prediction objective described below.

In practice, the training objective is often simplified by reparameterizing the loss to focus on predicting the noise ϵ that was added at each step. For sufficiently small β_t (which is typically ensured by proper noise schedule design), the KL divergence term in the ELBO can be approximated by a simple mean squared error between the true noise and the predicted noise:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{x_0,t,\epsilon}\left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, t)\|^2\right],$$

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ is the standard Gaussian noise and ϵ_θ is a neural network that estimates this noise component. This simplification, introduced by Ho et al. [HJA20], has been empirically shown to yield strong results while being more stable to train than the full ELBO objective.

Training proceeds by randomly sampling a time step t, corrupting x_0 according to the forward process to obtain x_t, and then optimizing the network to predict the noise component.

Despite their theoretical elegance and empirical success, diffusion models face challenges such as slow sampling (requiring T sequential denoising steps) and potential difficulties in modeling complex multimodal distributions.

Conditional generation modifications When adapting diffusion models for conditional sample generation, minor yet impactful modifications are introduced to incorporate auxiliary information c (e.g., class labels, textual descriptions) into the generative process. The key change is to modify the reverse process so that it models the conditional distribution:

$$p_\theta(x_{t-1} \mid x_t, c) = \mathcal{N}\Big(x_{t-1}; \mu_\theta(x_t, t, c),\, \Sigma_\theta(x_t, t, c)\Big).$$

This is typically achieved by embedding the conditioning variable c using an auxiliary network or learned embeddings and then integrating it into the noise prediction network. Common methods include concatenating c with the input x_t or leveraging cross-attention mechanisms, thereby allowing the network to incorporate the conditioning signal at each denoising step. The training objective is adjusted accordingly to minimize the KL divergence between the true posterior $q(x_{t-1} \mid x_t, x_0, c)$ and the conditioned reverse process $p_\theta(x_{t-1} \mid x_t, c)$.

Furthermore, the simplified noise prediction loss $\mathcal{L}_{\text{simple}}$ is also modified to incorporate conditioning:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{x_0,t,\epsilon}\left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, t, c)\|^2\right].$$

Here, the noise prediction network ϵ_θ is now conditioned on c, ensuring that the model learns to remove noise in a way that aligns with the conditioning information. This allows the generative process to produce samples that adhere to the desired attributes while maintaining the fundamental structure of diffusion-based denoising.

During sampling, techniques such as classifier guidance or classifier-free guidance [HS22] can be employed to steer the generation process more strongly towards the condition c, enabling a flexible trade-off between fidelity and conditioning adherence without altering the core diffusion framework.

2.4 Diffusion Models for Combinatorial Optimization

In this section, we review the state-of-the-art in applying diffusion models to CO. Specifically, we introduce the Difusco framework [SY23] and discuss subsequent works that build upon and refine its approach. Difusco represented the first diffusion-based general neural solver for CO problems.

2.4.1 Difusco

Difusco (DIffusion solvers for Combinatorial Optimization) is a supervised learning framework that utilizes graph-based diffusion models to address NP-hard combinatorial optimization (CO) problems, which are reformulated as node-classification or link-prediction tasks on graphs.

The fundamental concept is straightforward: similar to image denoising, the goal is to learn how to denoise randomly initialized graphs into high-quality feasible solutions of the respective CO problem. The model is trained by introducing noise to high-quality solutions on graphs and subsequently learning to reverse the noise, restoring the graphs to their original high-quality states. The paper conducts experiments on the TSP and MIS problems, which serve as examples of link-prediction and node-classification tasks, respectively. Figure 2.2 illustrates the denoising process applied to a TSP instance.

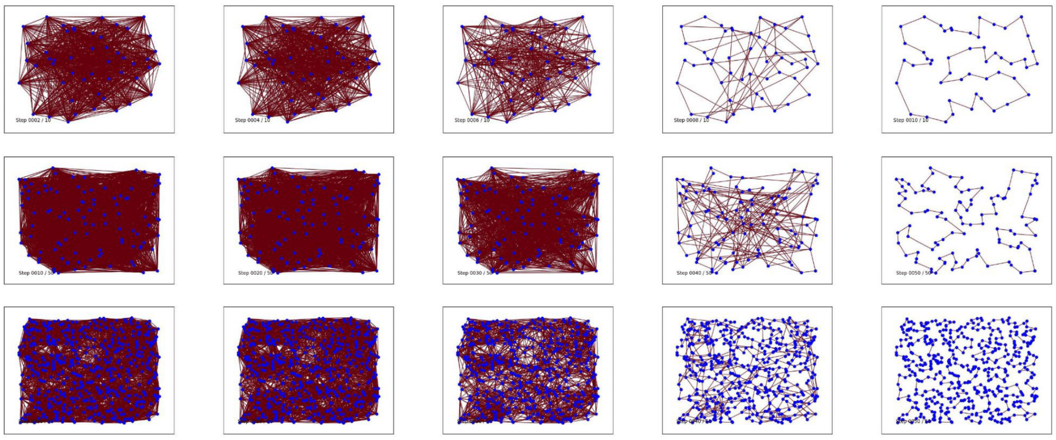

Figure 2.2: Difusco denoises a random-noise graph into a TSP solution. Figure taken from [SY23].

Architecture The paper investigates two diffusion paradigms: Gaussian diffusion and Bernoulli diffusion. Gaussian diffusion operates in continuous space by progressively adding Gaussian noise to solution vectors. Using the notation introduced in Section 2.3, the denoising process learns to recover clean solutions through $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$, where β_t controls the noise schedule. Bernoulli diffusion, on the other hand, works in a discrete state space using categorical transitions, defined as $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathrm{Cat}(\mathbf{x}_t; \mathbf{x}_{t-1}\mathbf{Q}_t)$, where $\mathbf{Q}_t$ is a transition matrix controlling bit-flip probabilities. The model learns to predict clean binary variables through discrete state transitions.

The denoising network is a 12-layer anisotropic GNN (introduced in Section 5.5) with edge gating mechanisms that handle both node and edge features. It is trained in a supervised manner by minimizing the negative log-likelihood of optimal solutions. During inference, Difusco employs adaptive cosine scheduling for noise levels to prioritize low-noise refinement and performs multi-step decoding, parallelizing the denoising of all variables in 50-200 steps, significantly fewer than traditional 1000-step diffusion processes.

Post-processing heuristics To handle constraint satisfaction and improve solution quality, Difusco incorporates post-processing heuristics. For the TSP, the decoding process is a greedy nearest-insertion heuristic biased by the predicted edge selection probabilities, as shown in Algorithm 2.3. The improvement heuristics used are 2-opt local search and Monte Carlo Tree Search (MCTS) guided by learned heatmaps as priors. For the MIS problem, the decoding process is a greedy node selection heuristic biased by the predicted node selection probabilities, as shown in Algorithm 2.4, and no improvement heuristics are used.

Results The discrete Difusco solver demonstrates significant improvements over previous neural solvers for CO problems. In the TSP across various scales, Difusco reduced the performance gap between ground-truth solutions and neural solver outputs employing Monte Carlo Tree Search (MCTS) decoding strategies. It showed very strong performance without the post-processing heuristics as well. Additionally, on the Maximal Independent Set (MIS) problem, Difusco outperformed existing neural solvers on the SATLIB dataset while achieving competitive results on the ER-[700-800] graphs using the Gaussian version of the diffusion model.

2.4.2 Subsequent Works

We now review the most relevant works that came after Difusco. We present (1) direct improvements and extensions to the Difusco framework, (2) works that explore the alternative paradigm of unsupervised diffusion learning, (3) works that hybridize diffusion models with other CO algorithms, and (4) works that apply diffusion models to other CO problems.

Direct improvements and extensions to Difusco Several works have built upon the Difusco framework to enhance its efficiency and scalability. [YZH+24] proposes DISCO, which improves diffusion-based solvers through residue-guided sampling and analytical denoising steps. By constraining the sampling space to solution residues while preserving multimodal distributions, DISCO achieves faster inference speeds without sacrificing solution quality, demonstrating 5.28× acceleration over baseline diffusion solvers on large-scale TSP instances. [HSY23] introduces progressive distillation to dramatically reduce the required denoising steps, compressing a 1000-step diffusion process into just 64 steps through iterative knowledge distillation. This approach maintains 99.98% of the original solution quality on TSP-50 while achieving 16× faster inference speeds.

Unsupervised diffusion learning and gradient-search paradigms Recent works have advanced unsupervised learning and alternative inference for diffusion-based CO. [LGWY23] introduces T2T, decoupling training from testing through supervised diffusion learning followed by unsupervised gradient-based search, achieving 49.15% improvement on TSP. [LZL+25] demonstrates zero-shot generalization across problem variants via energy-guided sampling with pretrained diffusion solvers. In purely unsupervised approaches, [HKJ+25] develops IC/DC, a constraint-aware diffusion model trained through self-supervision that achieves state-of-the-art on some benchmark problems without labeled optima. [SHL24] formalizes this paradigm using upper-bounded KL divergence for training-free constraint satisfaction through modified diffusion sampling.

Hybridization with other CO algorithms The recent work of [FSY24] embeds diffusion models within Large Neighborhood Search for ILP, where diffusion processes generate destruction patterns for the LNS repair phase. This neural-guided variant outperforms classical heuristics by 18.7% on MIS and Set Cover benchmarks, demonstrating how diffusion models can enhance traditional optimization through targeted subsystem replacement.

Applications in network optimization and beyond The versatility of diffusion-based CO has enabled its application across diverse domains. In network optimization, [LYC+24] and [LYY+25a] demonstrate how diffusion models can address complex network challenges, with the solver GDSG achieving near-100% task orthogonality in multiserver computation offloading. [LYY+25b] provides theoretical foundations for these applications, analyzing diffusion models' convergence properties in network optimization contexts.

Beyond networks, [LDDB24] adapts diffusion models for trajectory optimization in robotics, introducing a hybrid loss function that maintains constraint satisfaction during diffusion sampling. Similarly, [KDM+24] extends diffusion models to optimization with unknown constraints, using them as constrained samplers for black-box optimization problems in chemistry and materials science.

2.5 Evolutionary Algorithms

2.5.1 Introduction

Evolutionary algorithms (EAs) are computational optimization techniques inspired by biological evolution, drawing principles from Charles Darwin's theory of natural selection and Gregor Mendel's laws of heredity. The first references to this class of algorithms are found in the work of [Fog62], [FOW66], [Rec73], and [Hol75].

These algorithms simulate mechanisms such as mutation, recombination, and selection to evolve solutions iteratively. Over decades, EAs have expanded into diverse branches, including Genetic Programming, Differential Evolution, and more, addressing problems in optimization, machine learning, and engineering design.

EAs maintain a population of candidate solutions, iteratively improving them through cycles of selection, variation (recombination and mutation), and survival of the fittest. By mimicking natural evolution, EAs balance exploration (searching new regions) and exploitation (refining known solutions). The process begins with a randomized or heuristic-generated population, evaluates solutions using a fitness function, and iteratively applies genetic operators to evolve better solutions. The pseudocode for a general EA is shown in Algorithm 2.5.

2.5.2 Evolutionary Algorithm Components

The effectiveness of evolutionary algorithms depends on the specific implementation of their core components: selection, recombination, mutation, and replacement strategies. Each component plays a crucial role in balancing exploration and exploitation within the search space.

Selection

Selection determines which individuals contribute to the next generation. Common methods include Fitness Proportional (Roulette Wheel) selection, where individuals are chosen with probability proportional to their fitness. This method is prone to dominance by "super-individuals" in early stages. Rank-Based selection addresses this issue by ranking individuals, with selection probabilities depending on rank rather than raw fitness, which mitigates fitness scaling issues. Tournament Selection randomly iteratively selects k individuals and passes the best to the next generation, where larger k increases selection pressure. Elitism preserves top individuals unchanged to ensure monotonic improvement in the best solution quality across generations.

Recombination (Crossover)

Recombination combines traits from parents to create offspring. One-Point and Multi-Point Crossover split parent chromosomes at random points and are particularly effective for linear representations. Uniform Crossover, where each gene is randomly chosen from either parent, is ideal for non-sequential problems. Problem-specific operators include PMX (Partially Matched Crossover), which preserves absolute gene positions in permutations (e.g., TSP); OX (Order Crossover), which maintains relative order for sequencing problems; and Edge Recombination, which focuses on preserving edges in graph-based problems like TSP. The choice of recombination operator significantly impacts the algorithm's ability to effectively explore the solution space while preserving beneficial solution structures.

Mutation

Mutation introduces small, random changes to maintain diversity. For binary representations, mutation typically involves flipping bits with some probability. Real-Valued representations often use Gaussian noise or polynomial mutation to perturb continuous variables. For permutation problems like TSP, mutation operators include swap (exchanging two elements), insert (moving an element to a new position), or invert (reversing a subsequence). Adaptive Mutation adjusts the mutation rate dynamically based on population diversity, increasing rates when the population converges to prevent premature convergence and reducing rates when diversity is high to allow more focused exploitation.

Replacement Strategies

Replacement strategies determine how offspring replace existing individuals in the population. Generational Replacement completely replaces the entire population each iteration, creating a clear separation between consecutive generations. Steady-State replacement, in contrast, replaces only a few individuals at a time (typically the worst-fit members), allowing good solutions to persist longer in the population. Crowding and Fitness Sharing techniques preserve niche solutions to avoid premature convergence, maintaining diversity by considering the similarity between solutions when determining which individuals to replace. The replacement strategy significantly affects the balance between exploration and exploitation, with generational approaches typically favoring exploration and steady-state approaches emphasizing exploitation.

Other Considerations

Several advanced considerations further enhance EA performance. Constraint Handling techniques include penalty functions that reduce fitness for constraint violations, repair algorithms that transform infeasible solutions into feasible ones, and specialized operators like boundary-aware mutation that respect problem constraints. Hybridization, particularly combining EAs with local search in what are known as memetic algorithms, enhances exploitation by intensively improving promising solutions. Parameter Tuning remains critical for EA performance, with key factors including population size, mutation rate (p_m), and crossover rate (p_c). These parameters are increasingly tuned adaptively during the evolutionary process, responding to the algorithm's progress rather than remaining fixed throughout the search.

2.5.3 EAs for the Maximum Independent Set Problem

Evolutionary algorithms (EAs) provide effective approaches for solving the Maximum Independent Set Problem. We examine two distinct EA formulations that demonstrate promising results for this NP-hard combinatorial problem.

The work of [BK94] utilizes a permutation-based encoding scheme where chromosomes represent vertex orderings. This encoding guides a greedy decoder that sequentially adds vertices to the independent set following the permutation order while respecting independence constraints. A key innovation is the rearrange function, which prioritizes vertices in the current independent set to enhance search efficiency. The algorithm employs a two-point partially mapped crossover (PMX) to preserve vertex orderings and a vertex-swap mutation to maintain diversity. The fitness function is defined as $f(\alpha) = \text{Decode}(\alpha)^2 + 1$, where $\text{Decode}(\alpha)$ returns the independent set size derived from permutation α. The algorithm terminates after T_{stop} generations without improvement.

The work of [LSS96] implements a more direct binary encoding scheme where each bit indicates the presence (1) or absence (0) of a vertex in the solution. Its fitness function, $f(I) = \sum_{i=1}^{n} x_i - n \cdot \sum_{(i,j)\in E} x_i x_j$, rewards including more vertices while heavily penalizing independence constraint violations. The genetic operators include proportional selection with linear dynamic scaling, two-point crossover with rate $p_c = 0.6$, and bit-flip mutation with rate $p_m = 1/n$.

Algorithm 2.3: TSP Tour Construction via Edge Merging

```
Input: Coordinates coords, adjacency matrix $\mathbf{A}$ with edge probabilities
Output: A TSP tour and number of merge iterations
Step 1: Initialize;
dists $\leftarrow$ pairwise distances from coords;
$\mathbf{A} \leftarrow \mathbf{0}_{N \times N}$;
route_begin $\leftarrow [0, 1, \ldots, N-1]$;
route_end $\leftarrow [0, 1, \ldots, N-1]$;
Step 2: Sort edges;
sorted_edges $\leftarrow$ argsort($-\mathbf{A}/$dists);
Step 3: Merge edges;
merge_count $\leftarrow 0$;
foreach edge $\in$ sorted_edges do
    $(i, j) \leftarrow$ (edge$//N$, edge$\%N$);
    begin_i $\leftarrow$ route_begin$[i]$;
    end_i $\leftarrow$ route_end$[i]$;
    begin_j $\leftarrow$ route_begin$[j]$;
    end_j $\leftarrow$ route_end$[j]$;
    if begin_i = begin_j or $i \notin$ {begin_i, end_i} or $j \notin$ {begin_j, end_j} then
        continue;
    end
    $\mathbf{A}[i, j] \leftarrow 1$;
    $\mathbf{A}[j, i] \leftarrow 1$;
    merge_count $\leftarrow$ merge_count $+ 1$;
    Update route connections based on $(i, j)$;
    if merge_count $= N - 1$ then
        break;
    end
end
Step 4: Close the tour;
$\mathbf{A}$[final_end, final_begin] $\leftarrow 1$;
$\mathbf{A}$[final_begin, final_end] $\leftarrow 1$;
Step 5: Extract tour;
tour $\leftarrow [0]$;
current $\leftarrow 0$, prev $\leftarrow -1$;
while len(tour) $< N + 1$ do
    next $\leftarrow$ max(nonzero($\mathbf{A}$[current]));
    if prev $\geq 0$ then
        next $\leftarrow$ next[next $\neq$ prev]
    end
    ;
    tour.append(next);
    prev $\leftarrow$ current;
    current $\leftarrow$ next;
end
return tour, merge_iterations;
```

Algorithm 2.4: MIS Decoding Algorithm

```
    Input: Predictions $\mathbf{p}$, adjacency matrix $\mathbf{A}$
    Output: A feasible MIS solution
 1  Initialize solution vector $\mathbf{s} \leftarrow \mathbf{0}$;
 2  Sort nodes by descending prediction scores: $\mathbf{o} \leftarrow \text{argsort}(-\mathbf{p})$;
 3  foreach $i \in \mathbf{o}$ do
 4      if $\mathbf{s}[i] = -1$ then
 5          continue;
 6      end
 7      Mark neighbors as ineligible: $\mathbf{s}[\mathcal{N}(i)] \leftarrow -1$;
 8      Include node in solution: $\mathbf{s}[i] \leftarrow 1$;
 9  end
10  return $\mathbf{s} = 1$;
```

Algorithm 2.5: Evolutionary Algorithm

```
    Output: Best solution found
 1  Initialize population $P \leftarrow \text{RandomPopulation}()$;
 2  Evaluate fitness of all individuals in $P$;
 3  while termination condition not met do
 4      Select parents $P_{\text{parents}} \leftarrow \text{Select}(P)$;
 5      Apply recombination to create offspring $P_{\text{offspring}} \leftarrow \text{Recombine}(P_{\text{parents}})$;
 6      Mutate offspring $P_{\text{offspring}} \leftarrow \text{Mutate}(P_{\text{offspring}})$;
 7      Evaluate fitness of $P_{\text{offspring}}$;
 8      Replace population $P \leftarrow \text{Replace}(P, P_{\text{offspring}})$;
 9  end
10  return Best individual in $P$;
```

CHAPTER 3

Methodology

In this chapter, we present the methodology developed for integrating diffusion models with evolutionary algorithms, specifically the Diffusion-Based Evolutionary Algorithm (DEA). In Section 3.1, we first establish a baseline evolutionary algorithm for the Maximum Independent Set problem, detailing the problem encoding, feasibility heuristic, and traditional evolutionary operators. Section 3.2 introduces the design of the DEA, focusing on diffusion-based initialization and recombination operators that leverage pre-trained Difusco models. Finally, Section 3.3 describes the benchmarking datasets used to evaluate our approach.

3.1 A baseline EA for the MIS problem

We start by describing a baseline EA for the MIS problem. Our design draws inspiration from the literature on evolutionary algorithms for the MIS problem that we reviewed in Section 2.5.3. We adopt the direct binary encoding scheme similar to that used by [LSS96], where each bit indicates the presence or absence of a vertex in the solution. However, unlike their approach, which penalizes constraint violations in the fitness function, we maintain feasibility throughout the evolutionary process using a greedy decoding heuristic, a strategy that shares conceptual similarities with the decoder used in [BK94]'s permutation-based approach. Our elitist selection scheme and tournament selection are standard EA practices, while our crossover operator is specially designed to balance exploration and exploitation by producing one offspring that preserves common features between parents and another that enhances diversity. The following subsections detail each component of our baseline EA.

Problem encoding We encode a solution to the MIS problem as a binary vector $x \in \{0,1\}^n$, where $n = |V|$ is the number of vertices in the graph. Each element x_i indicates the presence (1) or absence (0) of vertex i in the independent set. We always keep the population of the EA feasible with respect to the independence constraint. The cost of a solution x is the size of the independent set, given by $f(x) = \sum_{i=1}^{n} x_i$.

Feasibility heuristic Different stages of the EA reuse the same feasibility heuristic to construct a feasible solution from a vector of node probabilities. We use the same heuristic as in Difusco [SY23], which we describe in Algorithm 2.4.

Initialization For each individual i in the population, where $i \in \{1, 2, \ldots, P\}$, we define $f_i = \mu \cdot i/P$ and $\mathbf{u}_i \sim \mathcal{U}(0,1)^n$, where μ is a hyperparameter that we set to 0.2. Then, the probability vector $\mathbf{p}_i \in [0,1]^n$ for individual i is defined as $\mathbf{p}_i = f_i \cdot \mathbf{u}_i$. We finally decode the probability vector $\mathbf{p}_i$ to obtain a feasible solution x_i using the feasibility heuristic.

Selection Given a population of P individuals, extended by P offspring, we use elitism to select the P best individuals that will be passed on to the next generation. This strategy corresponds to the Steady-State selection scheme. Since the algorithm is to be used in settings with a small population size, we experiment with the possibility of keeping only unique solutions in the population. In the case of the number of unique solutions being $P_u < P$, we sort the unique solutions by decreasing cost $c_1 \geq c_2 \geq \ldots \geq c_{P_u}$ and form the next generation by cyclically repeating the ordered set until a total of P solutions is reached.

Crossover The process of generating parent pairs utilizes tournament selection with a fixed size of $T = 2$. In evolutionary algorithms (EAs) with small population sizes, larger values of T can result in repeated pairs.

We employ a classical crossover operator that produces two offspring from two parents. Given two parents x and y, the crossover operator $(x, y) \mapsto (r, s)$ functions as follows:

1. Compute $U = \{i \in [n] \mid x_i = y_i = 1\}$, the set of common selected vertices of x and y.
2. Sample two vectors of probabilities $\mathbf{p}_1, \mathbf{p}_2 \sim \mathcal{U}(0,1)^n$.
3. Set $\mathbf{p}_1[i] = 1$ if $i \in U$, and $\mathbf{p}_2[i] = 0$ if $i \in U$.
4. Decode $\mathbf{p}_1$ and $\mathbf{p}_2$ to obtain two offspring r and s.

The crossover operator effectively combines both exploration and exploitation. The offspring r will resemble parents x and y because we set to one the selection probabilities of the vertices that are present in both x and y. In contrast, offspring s is designed to be as different as possible from x and y by setting to zero the selection probabilities of the vertices that are present in both x and y, thereby enhancing population diversity.

Mutation We mutate a solution with probability p_m. If a solution x is mutated, we will deselect each vertex selected in the current solution with probability p_d. We enforce the deselection by defining the mutated solution x' as the result of decoding a probability vector $\mathbf{p} \sim \mathcal{U}(0,1)^n$ with $\mathbf{p}[i] = 0$ if $x_i = 1$ and to be deselected, and $\mathbf{p}[i] = 1$ if $x_i = 1$ and not to be deselected.

3.2 Diffusion-Based Evolutionary Algorithm Design

We now describe the design of our proposed diffusion-based EA for the MIS problem. It is based on the diffusion-based initialization and recombination operators. Figure 3.1 provides an overview of the DEA architecture in comparison to a standard EA.

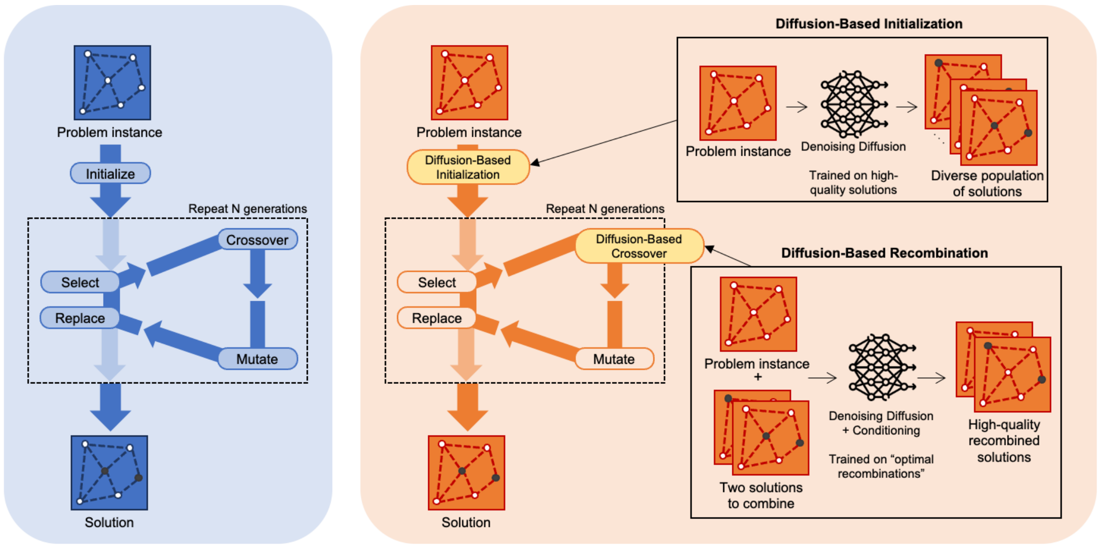
Problem instance
Initialize
Repeat N generations
Crossover
Select
Replace
Mutate
Solution
Problem instance
Diffusion-Based Initialization
Repeat N generations
Diffusion-Based Crossover
Select
Replace
Mutate
Solution
Diffusion-Based Initialization
Problem instance
Denoising Diffusion
Trained on high-quality solutions
Diverse population of solutions
Diffusion-Based Recombination
Problem instance +
Two solutions to combine
Denoising Diffusion + Conditioning
Trained on "optimal recombinations"
High-quality recombined solutions

Figure 3.1: Overview of the DEA design in contrast to the baseline EA.

Diffusion-based initialization The diffusion-based initialization is straightforward: we sample a population of P heatmaps using the pre-trained Difusco model on the problem distribution and decode them into solutions using the feasibility heuristic. Since we work with small population sizes, we use parallel sampling. In this step, we leverage the multimodal property of the Difusco model to sample a diverse population of high-quality solutions, which is a desired property for EA initialization.

Diffusion-based recombination The idea behind a diffusion-based recombination operator is to use a Difusco-inspired model to sample a high-quality offspring conditioned on the parents. In order to train the diffusion recombination model, we use imitation learning: the model learns from an expert demonstrator, namely the optimal recombination operator described in Section 3.2.1.

We assume that the population of P solutions is paired using tournament selection with a fixed size of $T = 2$. For each pair of parents, we sample two heatmaps from the conditioned diffusion recombination model, as detailed in Section 3.2.2. The inference step is parallelized across both the pairs and the two heatmaps. By conditioning on the two parents as additional node features, we aim for the model to effectively utilize the information from the parents to generate higher-quality offspring. The heatmaps are subsequently decoded using the feasibility heuristic to produce P offspring.

3.2.1 An Optimal Recombination as an Expert Demonstrator

As mentioned before, this operator is designed to be used as an expert demonstrator for the diffusion recombination operator. As such, it is designed with two considerations: (1) it is allowed to operate without strict runtime constraints, allowing for longer computation times when necessary, and (2) it should produce high-quality recombination results.

We base the operator on an ILP that we solve with an exact solver with a time limit. Inspired by local branching constraints in branch-and-bound algorithms [FL03], we significantly constrain the search space of the general MIS formulation presented in Example 2.1.4 with a local branching constraint. These constraints have the general form $h(x, x') \leq k$, where x is a fixed solution that we want to improve, x' is the decision vector, and h is a distance function. The hyperparameter k controls a trade-off: larger values expand the search space, potentially leading to higher-quality solutions but increasing computational cost; smaller values reduce the search space, enabling faster solving but potentially sacrificing solution quality.

Given parent solutions x and y, we aim to find a child solution z subject to (1) z must be an independent set, and (2) the combined Hamming distance between z and both parents x and y is bounded by k, where $k = \lambda \cdot h(x, y)$. λ is a hyperparameter that we will tune in Section 4.6. Let $G = (V, E)$ be a graph with vertex set V and edge set E, $x, y \in \{0, 1\}^n$ be binary vectors representing parent solutions, $z \in \{0, 1\}^n$ be the child solution (decision variables), and $h(a, b)$ denote the Hamming distance between two solutions a and b. We define $S_x = \{i \in V \mid x_i = 1\}$ and $U_x = V \setminus S_x$, as well as $S_y = \{i \in V \mid y_i = 1\}$ and $U_y = V \setminus S_y$. Then, the ILP is given by:

$$\begin{aligned}
\max \quad & \sum_{i \in V} z_i && && (3.1)\\
\text{s.t.} \quad & z_u + z_v \leq 1 && \forall (u,v) \in E && (3.2)\\
& \underbrace{\left(\sum_{i \in S_x} (1 - z_i) + \sum_{i \in U_x} z_i\right)}_{h(z,x)} + \underbrace{\left(\sum_{i \in S_y} (1 - z_i) + \sum_{i \in U_y} z_i\right)}_{h(z,y)} \leq k && && (3.3)\\
& k = \lambda \cdot \left(\sum_{i \in S_x} (1 - y_i) + \sum_{i \in U_x} y_i\right) && && (3.4)\\
& z_i \in \{0,1\} && \forall i \in V && (3.5)
\end{aligned}$$

We optimize the size of the independent set of the child with (3.1). Constraints (3.2) ensure that the child is an independent set. Constraint (3.3) ensures that the combined Hamming distance between the child and both parents is bounded by k. Constraint (3.4) defines k as a function of the Hamming distance between the parents. Finally, constraints (3.5) ensure that the child is a binary vector.

3.2.2 The Diffusion-Based Recombination Model

Here we describe the details of the diffusion model used for the diffusion-based recombination. In particular, the modifications we introduce to Difusco to allow for conditioning on extra features in the inference step.

Anisotropic GNN We first start by describing the anisotropic GNN used in Difusco. The formula for the message passing mechanism of this network can be found in 5.5. These networks generally perform well in both node- and edge-level prediction tasks. The specific architecture implemented in Difusco is the following:

- We first embed the input graph into a latent space using a sinusoidal positional embedding [VSP+17]. We note the embedding or hidden dimension as d_h.
- We then apply a linear transformation to the positional embeddings to obtain the initial node representations. This is implemented as a linear layer that maps from the embedding dimension to the same dimension (d_h).
- We apply a sequence of n_l anisotropic message passing layers based on the Gated Graph ConvNet design [BL18, DJL+22]. After that, the node features maintain the hidden dimension (d_h).
- For node probability generation, the final hidden representations are passed through an output sequence consisting of a normalization of the hidden features, a ReLU

activation, and a 1x1 convolution that transforms from hidden dimension to a single output channel, which becomes the node probability.

Modifications for conditioning on extra node features In order to condition the diffusion model on extra node features, we modify the embedding modules of the model. Assuming x is a sample graph with n nodes and m features (previously $m = 1$), we apply the following modifications:

- We embed each feature vector x_i independently using a sinusoidal positional embedding of dimension d_h.
- We add a linear transformation layer for each embedded feature vector, which maps from the embedding dimension to the hidden dimension (d_h).
- As common in the literature (e.g. [VSP+17]), we sum the transformed embedded feature vectors to get a d_h-dimensional vector for each node.

As explained in Section 2.3, conditioning a diffusion process on extra features is a standard practice. The changes in the training regime are minimal; the loss function is essentially the same as in the unconditional case.

Number of model parameters Using the architecture described above with the hyperparameters $d_h = 256$ and $n_l = 12$, the number of parameters of the original Difusco model is 5,333,505. After conditioning on the extra node features, the number of parameters is increased by $(m-1) \cdot (d_h^2 + d_h)$. In our case of $m = 3$, we reach a network of size 5,465,089.

3.3 Benchmarking Datasets

As with much of the literature on neural solvers [SY23, YZH+24, LGWY23, LZL+25, SBK22], we focus on the Maximum Independent Set problem. We additionally perform some of our experiments on the Traveling Salesperson Problem.

3.3.1 Maximum Independent Set

We use the following benchmark datasets:

- **Erdős-Rényi graphs**: We generate Erdős-Rényi random graphs by setting the edge probability to $p = 0.15$ using the framework provided by [BKT+22]. We create datasets of different sizes: $50 \leq n \leq 100$, $300 \leq n \leq 400$, $700 \leq n \leq 800$, and $1300 \leq n \leq 1400$. A total of 40,000 samples are produced for each dataset, with 128 defined as the test set and the remainder as the training set. For the 700-800 dataset, we use exactly the same test instances as in the original Difusco paper, ensuring a fair comparison. For the 1300-1400 dataset, we only consider a 128-instance test set, since it will be used for out-of-distribution generalization experiments. Labeling the instances is done using the KaMIS heuristic solver [LSS+17] with a time limit of 60 seconds.

- **SATLIB**: Moreover, we use the SATLIB [HS00] benchmark for MIS, which is a translation of SAT instances into MIS instances. We use the same test instances as in the original Difusco paper, ensuring a fair comparison. We do not generate any training instances, since we do not work on this dataset for the diffusion-recombination experiments. The graph sizes in the dataset range from $1200 \leq n \leq 1400$.

3.3.2 Traveling Salesperson Problem

We reuse the datasets from the original Difusco paper, which are labeled using the Concorde TSP solver [ABCC06] and the LKH-3 solver [Hel00]. We summarize them in Table 3.1.

Table 3.1: Overview of datasets and solvers used for different graph sizes.

Number of Nodes	Test Set Size	Solver Used
50	1280	Concorde
100	1280	Concorde
500	128	LKH-3
1000	128	LKH-3
10000	16	LKH-3

CHAPTER 4

Computational Experiments

In this chapter, we present the computational experiments of the thesis. We first outline the implementation details and the computational resources used for the experiments in Section 4.1. We then proceed with the experiments and results. In Section 4.2, we investigate the importance of post-processing heuristics in Difusco's performance on TSP problems. Section 4.4 examines whether Difusco exhibits the multimodal property that makes it suitable for population-based approaches, analyzing both TSP and MIS problems. In Section 4.5, we study how Difusco's runtime scales with problem size and batch processing, assessing the efficiency of parallel inference on different datasets. Section 4.9.3 evaluates our DEA framework against a Naïve EA baseline and conducts ablation studies to understand the contribution of diffusion-based components. Section 4.10 provides a comprehensive comparison of our DEA framework against other state-of-the-art solvers, including Difusco, Gurobi, and KaMIS across three datasets. Finally, Section 4.11 examines the out-of-distribution generalization capabilities of both Difusco and DEA on significantly larger graph instances beyond the training distribution. Throughout these sections, we progressively build evidence for the efficacy of our hybrid approach that integrates diffusion models with evolutionary algorithms.

4.1 Implementation Details

We build upon the implementation of Difusco [SY23]. The implementation is done in Python and leverages PyTorch [PGC+19] for neural network operations, PyTorch Lightning [Fal19] for training organization, and PyTorch Geometric [FL19] for GNNs and graph-related operations. The EA is implemented using EvoTorch [TAM+23], a library for building and running EAs with PyTorch. We solve the problems of the optimal recombination operator using Gurobi 12.0 [GO25] as the ILP solver.

If not otherwise specified, the experiments that involve Difusco/DEA inference or training are run on a single NVIDIA A100 80GB (SXM4) GPU. The rest of the experiments are run on a cluster with an Intel(R) Xeon(R) E5-2640 v4 CPU with 2.40GHz and 160GB RAM, running Ubuntu 18.04.6 LTS.

The code, checkpoints for the trained models, and the test instances are available at `https://github.com/jsalvasoler/difusco-dea` under the MIT license.

4.2 Study of the Effect of the Post-Processing TSP Heuristics in Difusco

A key goal of this thesis is to develop a general approach that avoids problem-specific heuristics. We start by investigating the impact of post-processing TSP heuristics used in Difusco [SY23], a study absent in the original work. If the effect were significant, this would further motivate our contribution of a problem-independent framework. We do not perform the experiment in the MIS problem because no post-processing heuristics for this problem were used in the original paper.

The setup of the experiment is as follows: we generate initial solutions using the greedy decoding of the paper, but starting with random noise. 1x means that we consider only one solution, while 16x means that we consider 16 solutions, i.e., we decode 16 random noise vectors. We then run the 2-opt heuristic on each of the solutions in the population. The maximum number of 2-opt improving moves is set to 5000, as in the paper. The Difusco results are taken from the paper. MCTS is not included in the table as it was not easily reproducible, and 2-opt experiments for the 10000-node graphs were not run as they were computationally very demanding. Empty cells in the table (marked as "–") represent experiments that were either not conducted or where data was not available from the original paper.

Table 4.1 presents the performance comparison of TSP solvers with and without Difusco across different graph sizes (50, 100, 500, 1000, and 10000 nodes). The initial solution column indicates the source: GD means greedy decoding of random noise, while Difusco means decoding of diffusion heatmaps. The method column shows the post-processing approach: xn means a population of n solutions with the best one reported. All results are averaged over three runs.

The insights from the table are the following:

- For small TSP instances (50, 100): Difusco methods (even without 2-opt) deliver near-optimal solutions, outperforming raw greedy approaches.
- For larger instances (500, 1000, 10000): Difusco without 2-opt is worse than greedy + 2-opt, which might be interpreted as a disappointing result. However, applying 2-opt to the Difusco heatmaps delivers higher quality solutions than greedy + 2-opt.
- The multimodal property of the Difusco heatmaps is evident: only decoding 16 solutions instead of one is enough to see a significant improvement. This effect is also enhanced by the 2-opt heuristic.

Table 4.1: Average TSP tour length for different Difusco post-processing settings across different graph sizes.

Method	Initial Solution	Dataset (Number of Nodes)				
		50	100	500	1000	10000
Optimal Labels	–	5.69	7.76	16.58	23.23	–
x1	GD	6.65	9.14	19.47	27.03	82.49
x16	GD	6.64	9.14	19.47	27.03	82.49
x1 + 2-opt	GD	5.86	8.03	17.17	24.02	–
x16 + 2-opt	GD	5.73	7.94	17.17	–	–
x1	Difusco	5.70	7.78	18.35	26.24	98.15
x16	Difusco	5.69	7.76	17.23	25.19	95.52
x1 + 2-opt	Difusco	–	–	16.80	23.56	73.99
x16 + 2-opt	Difusco	–	–	16.65	23.45	73.89
MCTS	Difusco	–	–	16.63	23.39	73.62

Therefore, we conclude that post-processing heuristics are critical for the performance of the Difusco method. Without them, Difusco is not better than a simple, general, and faster greedy approach. However, when paired with the 2-opt heuristic or MCTS, Difusco delivers very high-quality solutions that require advanced solvers to beat.

4.3 Training Difusco for Additional MIS Datasets

To expand the evaluation scope of our approach, we trained Difusco models for two additional datasets: ER-50-100 and ER-300-400. For the ER-700-800 and SATLIB instances, we utilized the pre-trained checkpoints provided by the original Difusco authors [SY23].

We adopted the following hyperparameter configuration for training: a batch size of 64, an initial learning rate of 0.0002, a weight decay coefficient of 0.0001, and a cosine-decay learning rate scheduler [LH17]. Following the recommendations in the original work, we set the number of diffusion steps to 1000 during training, while using 50 steps for inference, and trained each model for 50 epochs. The training is performed on a single NVIDIA A100 80GB (SXM4) GPU.

Figure 4.1 shows the training evolution for the ER-50-100 dataset, illustrating the convergence behavior of the loss function over epochs.

Table 4.2 summarizes the single-sample or one-shot performance of the trained models on the validation set. Test GT Cost is the average cost of the optimal labels, Test Cost indicates the average cost of the one-shot Difusco inference, and Test Greedy Cost shows the average cost of a simple greedy decoder. The Training Error column reports the average loss over the final 100 training steps.

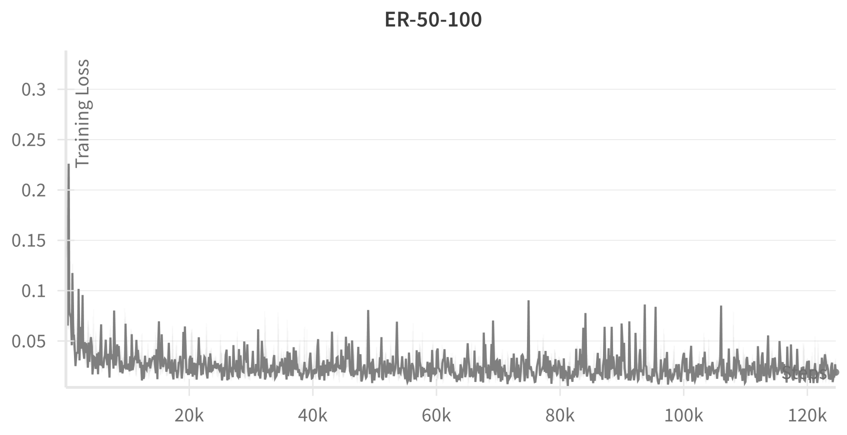

Figure 4.1: Training loss evolution of the Difusco model on the ER-50-100 dataset across 50 epochs.

Table 4.2: Comparison of one-shot inference performance on validation sets and training error for the trained Difusco models.

Dataset	Test GT Cost	Test Cost	Test Greedy Cost	Training Error
ER-50-100	20.75	18.16	15.8984	0.01337
ER-300-400	37.06	25.96	25.1797	0.01258

Both models achieve comparable convergence in terms of training error. Difusco shows strong one-shot performance on the ER-50-100 dataset, significantly outperforming the greedy decoder and achieving a reasonable 12.5% gap to the optimal solution. In contrast, the performance on the ER-300-400 dataset is less impressive, with Difusco only marginally surpassing the greedy decoder and exhibiting a substantial 30.0% optimality gap.

It is worth noting, however, that one-shot performance metrics do not fully capture Difusco's capabilities. The model's true strength lies in its ability to generate diverse, high-quality solutions through multiple samplings—a property we exploit in our evolutionary framework. As demonstrated in subsequent sections, the trained models exhibit competitive performance when sampling multiple solutions to form a population.

4.4 Validation of the Multimodal Property of the Difusco Heatmaps

We have already discussed the promised multimodal property of Difusco: by leveraging the stochastic denoising process of diffusion models, the framework generates distinct high-quality solutions when initialized with different noise samples. This property enables the model to capture a diverse, multimodal distribution of near-optimal solutions, addressing a critical limitation of deterministic or unimodal neural solvers.

There is evidence for this property in the results of the original paper, and also in the previous Section 4.2. We now proceed with a more systematic analysis of the multimodal property. The results of this section will also be used to compare against the EA framework because the initialization of the DEA essentially samples a population of solutions. This is what we will later refer to as the initialization gap.

For the experiment, we sample P times Difusco, where $P \in \{4, 8, 16, 32\}$, and apply simple greedy decoding to each of the solutions. We use the same hyperparameters as in the original paper, which are described in Section 5.5. We apply no post-processing heuristics. The metric we measure is the gap with respect to the optimal label, defined differently for the TSP and MIS problems to account for the optimization sense:

$$\text{gap (TSP)} = \left(\frac{\text{tour length} - \text{optimal tour length}}{\text{optimal tour length}} \right) \times 100\,(\%) \tag{4.1}$$

$$\text{gap (MIS)} = \left(\frac{\text{optimal IS size} - \text{IS size}}{\text{optimal IS size}} \right) \times 100\,(\%) \tag{4.2}$$

The results are shown in Figure 4.2 for the MIS problem, and in Figure 4.3 for the TSP problem.

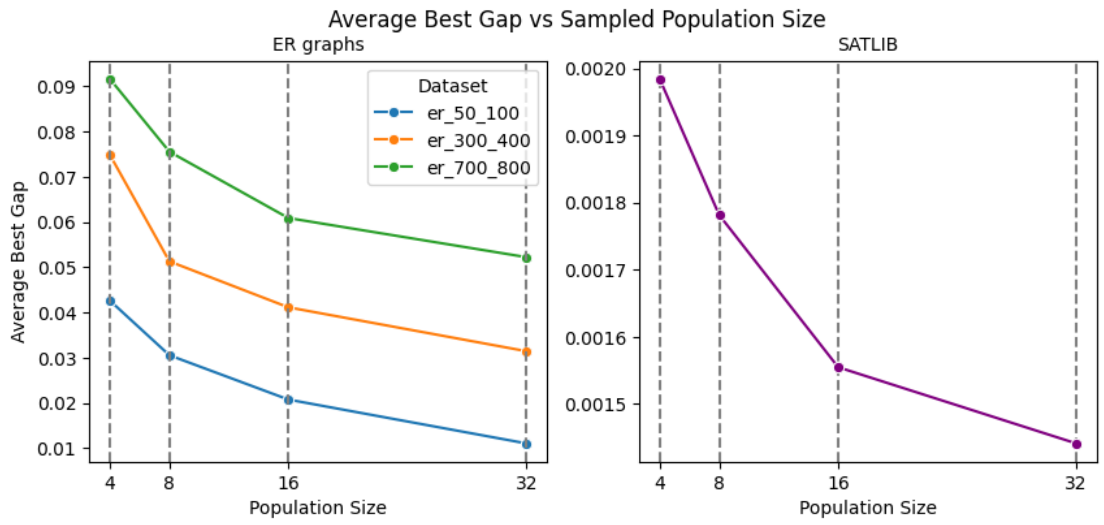

Figure 4.2: Average optimality gap (lower is better) of the best solution in a population of Difusco-sampled solutions as a function of the population size. As the population size increases, the gap to the optimal solution decreases, demonstrating the multimodal nature of Difusco's solution distribution. We plot ER graphs and SATLIB instances separately.

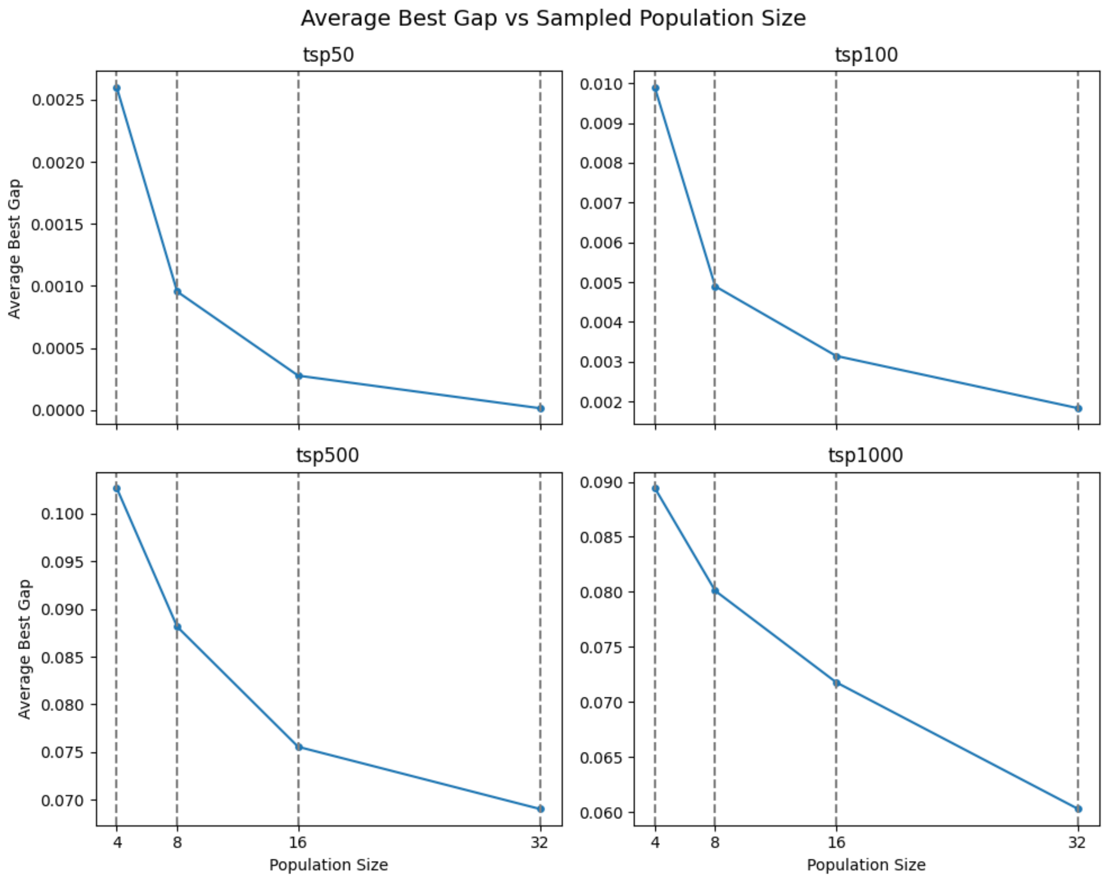

Figure 4.3: Average optimality gap (lower is better) of the best solution in a population of Difusco-sampled solutions as a function of the population size for the TSP problem. The decreasing curve demonstrates that sampling multiple solutions from Difusco yields progressively better solutions, confirming the multimodal property.

We clearly see monotonically decreasing curves, which indicates that the multimodal property is at play. We also observe a diminishing returns effect, where the gap with respect to the optimal label decreases with the number of Difusco samples, but the improvement in gap is smaller for larger P. This property is not surprising, as the gaps are bounded by the quality of the true optimal solution. A linear behavior instead of the observed asymptotic decrease could only happen when the solution pools are very far from the optimal solution, which does not seem to be the case as observed in the small gap values. For the ER-300-400 dataset, which we trained ourselves, we observe that the high one-shot gap values noted in Section 4.3 remain evident in the 4-sample case. However, as the number of samples increases, the gap decreases rapidly, ultimately achieving results comparable to those of the ER-50-100 and ER-700-800 datasets.

4.5 Runtime Scaling of Difusco

One of the strengths of our DEA approach is that it has many parallelizable components and can leverage GPU-accelerated operations. We study the runtime scaling of two critical components: the greedy decoding, which is used at every operator of the EA, and the parallel Difusco inference, which is used at the initialization of the EA and in the diffusion-based recombination.

4.5.1 Runtime Scaling of the Greedy Decoding

We study the runtime scaling of the greedy decoding as a function of the batch size. We benchmark two implementations:

- A NumPy implementation of the greedy decoding that relies on `scipy.sparse` matrix operations for fast neighborhood queries. This implementation processes a batch of heatmaps sequentially, meaning that we cannot expect better than linear scaling. We call it NumPy (sequential).

- A PyTorch implementation of the greedy decoding that uses precomputed neighborhood maps for each node to exploit sparsity. This implementation processes a batch of heatmaps in a vectorized manner, meaning that we can expect better than linear scaling. We call it PyTorch (vectorized).

No fair comparison can be made between the two implementations, since one is vectorized and the other one is not. However, we provide the results for both implementations for completeness.

We compare decoding performance across different batch sizes and hardware setups (CPU vs. CUDA). In PyTorch, CUDA indicates that tensors reside in GPU memory and operations are executed on the GPU. For NumPy, CUDA means that tensors are first transferred from the GPU to the CPU, converted to NumPy arrays, and then processed. This simulates a scenario where the NumPy implementation may be faster, making it advantageous to run the greedy decoding on the CPU. The runtimes in milliseconds of decoding a solution, averaged over 100 runs, are shown in Table 4.3.

The results indicate that the vectorized PyTorch implementation exhibits strongly sublinear scaling on both CPU and CUDA devices, confirming the success of vectorization. In contrast, the NumPy implementation demonstrates linear scaling, as expected for a sequential algorithm. Overall, both implementations run significantly faster on the CPU than on CUDA. This discrepancy arises from the overhead associated with GPU indexing and slicing operations. Additionally, the greedy algorithm processes nodes sequentially, which inherently increases the number of operations and execution steps, further amplifying GPU overhead.

Table 4.3: Runtime comparison (in milliseconds) of greedy decoding across different batch sizes and for different datasets.

			Batch Size			
Dataset	Device	Implementation	16	32	64	128
ER-50-100	CPU	NumPy	19.3	33.5	70.8	136.9
		PyTorch	4.7	4.0	5.9	5.1
	CUDA	NumPy	17.4	39.3	73.7	119.5
		PyTorch	17.7	24.2	26.1	22.8
ER-300-400	CPU	NumPy	29.3	59.7	116.4	240.7
		PyTorch	12.8	17.1	24.2	22.4
	CUDA	NumPy	31.0	59.4	122.4	234.1
		PyTorch	62.5	88.8	98.1	94.1
ER-700-800	CPU	NumPy	36.1	68.7	140.5	294.6
		PyTorch	23.5	29.4	38.2	50.7
	CUDA	NumPy	36.3	70.8	143.4	290.3
		PyTorch	109.0	136.9	168.9	204.1

4.5.2 Runtime Scaling of Parallel Difusco Inference

Another scaling study we perform is for the parallel Difusco inference. Typically, batched inference runtimes scale sublinearly when the batch sizes increase linearly. This is due to the parallelization capabilities of the GPU. For this purpose, we measure the scaling of the parallel Difusco inference for different batch sizes. This study is performed on a single NVIDIA GeForce RTX 3090 GPU, although we observed similar scaling behavior on the A100 GPU. Figure 4.4 shows the runtimes in milliseconds of sampling a batch of heatmaps in parallel, averaged over five runs.

We only observe the expected behavior for the ER-50-100 dataset, where the runtime scales strongly sublinearly. For ER-300-400, it scales slightly better than linearly, and ER-700-800 is essentially linear.

We hypothesize that this degradation in scaling efficiency may stem from the computational overhead of multiple sequential denoising steps in Difusco's inference process. A more lightweight implementation could potentially mitigate this issue and improve runtime performance.

4.6 Experiments on the Optimal Recombination Operator

As explained in Section 3.2.1, we design a recombination operator for the MIS with the following considerations: (1) it should produce high-quality offspring that combine beneficial traits from both parents, (2) it can operate without strict runtime constraints,

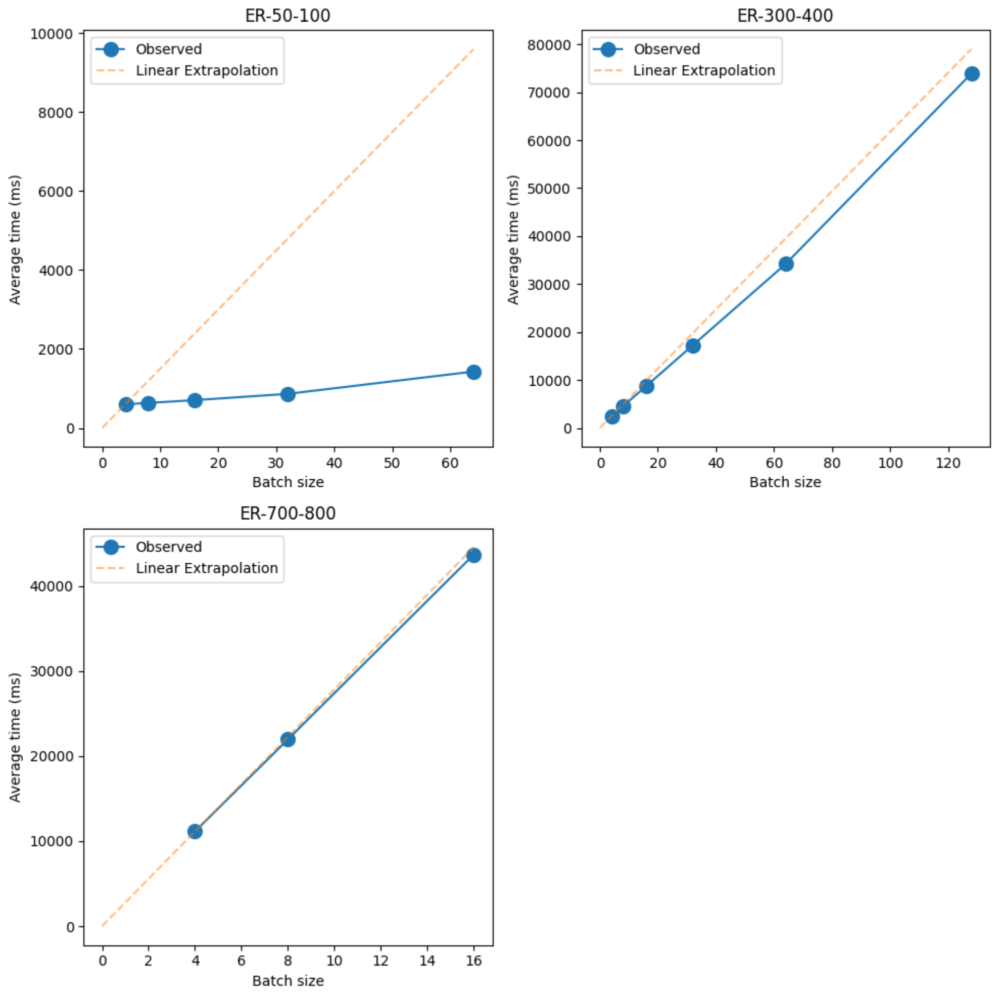

Figure 4.4: Runtime comparison (in milliseconds) of parallel Difusco inference across different batch sizes and for different datasets.

allowing for longer computation times when necessary, and (3) it should serve as an expert demonstrator for training the diffusion recombination operator.

In Section 3.2.1, we presented the final recombination operator, which consists of solving the ILP formulation in equations 3.1-3.5. Here, we perform comparative studies with two other recombination operators that confirm the choice of this design and determine the optimal value of the hyperparameter λ.

4.6.1 Alternative Recombination Approaches

Before settling on the local branching-based recombination operator, we explored two alternative approaches:

Weighted MIS This approach formulates a weighted MIS problem where solutions similar to the parents are rewarded through the objective $\sum_{i \in V} w_i z_i$. Let $S_x = \{i \in V \mid x_i = 1\}, S_y = \{i \in V \mid y_i = 1\}$ be the set of selected nodes in the parent solutions x and y, and $\theta \in [0, 1]$ a penalty parameter. We define the weight of node i as:

$$w_i = \begin{cases} 1+\theta & \text{if } i \in S_x \cap S_y \\ 1 & \text{if } i \in S_x \setminus S_y \text{ or } i \in S_y \setminus S_x \\ 1-\theta & \text{if } i \notin S_x \cup S_y \end{cases} \tag{4.3}$$

Constrained MIS In this other approach, instead of rewarding solutions similar to the parents through the objective, we can constrain the search space around the parents. We can do this by fixing the selection of nodes that are present in both parent solutions or excluding a selection of nodes that are not selected in both parents. A third strategy is to do both. In this case, the problem can be formulated as:

$$\begin{aligned} \max \quad & \sum_{i \in V} z_i && && (4.4) \\ \text{s.t.} \quad & z_u + z_v \leq 1 && \forall (u, v) \in E, && (4.5) \\ & z_i \geq \alpha_{\text{select}} && \forall i \in S_x \cap S_y, && (4.6) \\ & z_i \leq 1 - \alpha_{\text{exclude}} && \forall i \notin S_x \cup S_y, && (4.7) \\ & z_i \in \{0, 1\} && \forall i \in V. && (4.8) \end{aligned}$$

The objective (4.4) maximizes the size of the independent set. Constraint (4.5) ensures that the solution is an independent set. Constraint (4.6) controls whether nodes present in both parent solutions ($S_x \cap S_y$) must be included in the offspring, with $\alpha_{\text{select}} = 1$ forcing their inclusion. Constraint (4.7) determines whether nodes absent from both parent solutions ($\notin S_x \cup S_y$) must be excluded from the offspring, with $\alpha_{\text{exclude}} = 1$ preventing their selection. Finally, constraint (4.8) ensures that z is a binary vector.

The three versions of this approach are obtained by setting the values of α_{select} and α_{exclude} as follows:

- $\alpha_{\text{select}} = 1$ and $\alpha_{\text{exclude}} = 0$: we fix the selection of nodes that are present in both parent solutions.
- $\alpha_{\text{select}} = 0$ and $\alpha_{\text{exclude}} = 1$: we only exclude the nodes that are not selected in any of the parents.
- $\alpha_{\text{select}} = 1$ and $\alpha_{\text{exclude}} = 1$: we fix the selection of nodes that are present in both parent solutions and exclude the nodes that are not selected in any of the parents.

4.6.2 Results and Tuning

The experiments for the optimal recombination operator are performed on a set of 15 instances from each dataset. For each instance, we sample 16 heatmaps using Difusco, decode using the feasibility heuristic, pair the solutions randomly, and solve using Gurobi with a time limit of 15 seconds. This creates a second set of 8 solutions. We report the average generation improvement as:

$$\text{Gen Improvement} = \frac{\text{Best parent objective} - \text{Best offspring objective}}{\text{Best parent objective}} \times 100\ (\%)$$

Unsatisfactory results for weighted MIS We first test the weighted MIS approach with different penalty values: $\theta \in \{0.05, 0.1, 0.5\}$. The results are shown in Table 4.4.

Table 4.4: Performance of the weighted MIS recombination operator measured by the average generation improvement with different penalty values.

Dataset	θ Penalty	Gen Improvement (%)
ER-50-100	0.05	**3.12**
	0.10	3.12
	0.50	1.13
ER-300-400	0.05	**3.58**
	0.10	3.03
	0.50	0.15
ER-700-800	0.05	0.00
	0.10	0.00
	0.50	0.00

The results show that this approach only works well with very small penalty values (which is essentially equivalent to solving the standard MIS problem without biasing the solution towards the parents), making the approach not very useful. On the ER-700-800 datasets, the recombination struggles to find any improvement. Due to this inability to find any improvement in one of the datasets, we discard this approach.

Tuning the configuration of Constrained MIS Next, we evaluate the constrained MIS approach with different configurations for fixing node selections and unselections. Table 4.5 presents the generation improvement across all datasets with the three different configurations. The optimal configuration involves excluding only the nodes that are not selected in either parent ($\lambda_{\text{select}} = 0$, $\lambda_{\text{exclude}} = 1$). Fixing the selection of nodes present in both parents alone seems to leave excessive freedom, which the solver cannot effectively exploit within the time limit. Conversely, combining both constraints overly restricts the search space, and solutions that improve the parents cannot be found.

Tuning the λ parameter for Local Branching We finally test the local branching approach described in Section 3.2.1, which constrains the combined Hamming distance between the offspring and both parents. The key hyperparameter to tune in this approach

Table 4.5: Performance of constrained MIS measured by the average generation improvement with different configurations regarding node selections and unselections.

Dataset	α_{select}	α_{exclude}	Gen Improvement (%)
ER-50-100	1	0	0.46
	0	1	**1.89**
	1	1	0.46
ER-300-400	1	0	0.00
	0	1	**3.28**
	1	1	0.00
ER-700-800	1	0	0.00
	0	1	**3.41**
	1	1	0.00

is λ, which determines the size of the search space. To find the optimal value of λ, we conducted experiments with different values ranging from 1.25 to 5.0, solving the resulting ILP with a time limit of 15 seconds. The results are presented in Figure 4.5.

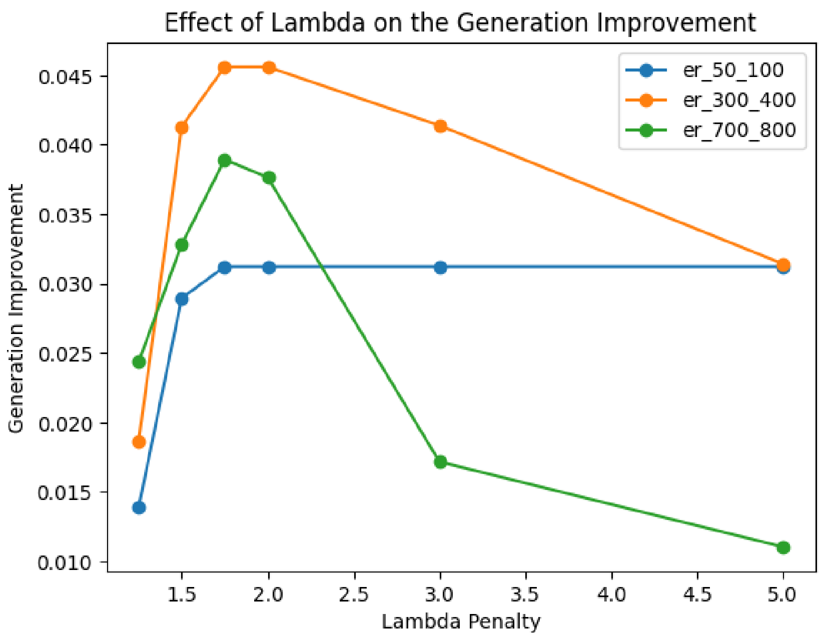

Figure 4.5: Impact of λ on the performance of the local branching recombination operator.

The plot clearly captures the exploration/exploitation trade-off. For small values of λ, the search space is too restricted, limiting the potential for improvement. For large values of λ, the search space becomes too large, making it difficult to find good solutions within the time limit. For all three datasets, the ideal value of λ was found to be around

1.75. For ER-50-100, values over 1.75 do not yield any improvement because the optimal solution of the MIS problem instance is found.

For the optimal value of $\lambda = 1.75$, we observed that the solver did not always reach optimality within the time limit for the larger datasets (ER-300-400 and ER-700-800), suggesting that allowing more time could potentially lead to even better solutions.

4.6.3 Final Comparison

Finally, we compare the performance of the local branching approach with the constrained MIS approach. Table 4.6 presents the results across all datasets, showing both the average generation improvement and the average runtime in seconds for each method. The constraint MIS uses $\alpha_{\text{select}} = 0$ and $\alpha_{\text{exclude}} = 1$, while the local branching approach uses $\lambda = 1.75$ as determined from our parameter tuning experiments.

Table 4.6: Comparison of constrained MIS and local branching recombination operators.

Dataset	Recombination	Gen Improvement (%)	Runtime (s)
ER-50-100	Constrained MIS	1.89%	0.03
	Local Branching	**3.12%**	0.19
ER-300-400	Constrained MIS	3.28%	17.84
	Local Branching	**4.56%**	27.91
ER-700-800	Constrained MIS	3.41%	29.45
	Local Branching	**3.89%**	30.61

The results clearly show that the local branching approach consistently outperforms the constrained MIS approach across all datasets, albeit with slightly longer runtimes. Based on these results, we selected the local branching approach with $\lambda = 1.75$ to be the optimal recombination operator acting as an expert demonstrator for the diffusion recombination model.

4.7 Evaluation of the Baseline Evolutionary Algorithm

In this section, we evaluate the performance of the baseline EA, comparing different settings.

Due to the computational cost of the Difusco sampling, which will need to be used at every recombination event, the population size is limited to $P = 16$. According to Figure 4.2, we benefit significantly from a population size of 16 instead of 4 or 8. Larger population sizes have diminishing returns. In this setting of a very small population size, it makes sense to modify the selection scheme to keep only unique solutions in the population. This modification is explained in Section 3.1.

Here, we perform computational experiments to benchmark the baseline EA with $P = 16, G = 20$, initialized with Difusco sampling. We compare unique vs non-unique populations with the classical vs optimal recombination operators. We report in Table 4.7 the gap of the final best solution to the optimal solution, along with the improvement with respect to the initialization gap of the population.

Table 4.7: Comparison of unique vs. non-unique population strategies, and classic vs optimal recombination operators. Darker blue cells indicate better performance.

Dataset	Unique Pop	Gap Improv.		Final Gap (%)		Initialization Gap (%)
		Classic	Opt.	Classic	Opt.	
ER-50-100	False	-1.17	-2.07	0.91	0.00	2.07
	True	-1.45	-2.07	0.62	0.00	2.07
ER-300-400	False	-0.23	-3.30	3.89	0.81	4.11
	True	-0.39	-3.53	3.72	0.58	4.11
ER-700-800	False	0.07	-2.61	6.16	3.48	6.09
	True	0.09	-2.88	6.18	3.21	6.09

The results indicate that maintaining unique populations generally leads to better performance, particularly with optimal recombination. From this table, we also conclude that the optimal recombination operator is effective at improving the initial population quality. We observe reductions in the initialization gap of -2.07, -2.53, and -2.88 for the ER-50-100, ER-300-400, and ER-700-800 datasets, respectively.

4.8 Training the Diffusion Recombination Model

Having established the effectiveness of our optimal recombination operator, we now use it to generate training data for our diffusion recombination model. The goal is to train a model that can quickly generate high-quality offspring without the computational overhead of solving an ILP.

4.8.1 Synthetic Data Generation

We generate synthetic training examples for the diffusion recombination model by running the EA with our optimal local branching recombination operator. By training on data from realistic evolutionary trajectories, we maximize the model's effectiveness when integrated into the complete EA framework, enabling it to generalize well to the types of parent solutions it will encounter during operation.

This data generation process is computationally expensive, as it requires solving an ILP with a mean runtime of T_{ILP} for each pair of parent solutions ($P/2$ pairs, where P is the population size), for each generation G of the EA, across N_{train} training instances. The total computational cost is approximately $T_{\text{ILP}} \cdot P/2 \cdot G \cdot N_{\text{train}}$. For the three datasets, we use the parameters shown in Table 4.8.

Table 4.8: Parameters used for synthetic data generation across different datasets and the resulting computational cost.

Dataset	P	G	T_{ILP} (s)	N_{train}	CPU Time (days)
ER-50-100	16	20	0.2	39,872	2.2
ER-300-400	16	3	15.1	33,750	140.6
ER-700-800	16	3	15.1	19,990	83.3

To manage this computational burden, we parallelize the generation process across different instances of the same dataset. Additionally, we implement a caching mechanism to store and reuse ILP solutions when the same parent pairs are encountered. The cache hit rate—the percentage of times a previously solved problem can be reused—is approximately 5% for the ER-300-400 dataset and 3% for the ER-700-800 dataset, providing only modest time savings.

Creation of the diffusion recombination dataset Our goal is to create a dataset of $N_{\text{rec-train}}$ recombination examples, where each example consists of a pair of parent solutions and their corresponding offspring solution. Multiple recombination examples can share the same underlying graph instance, meaning $N_{\text{train}} < N_{\text{rec-train}}$. To ensure dataset quality, we apply the following filtering criteria to the collected examples $(x, y; z)$:

- Exclude symmetric duplicates where $(x, y; z) = (y, x; z)$
- Exclude examples with identical parents $(x = y)$
- Exclude examples where the offspring is identical to either parent ($z = x$ or $z = y$)

From the filtered set, we sample $N_{\text{rec-train}}$ examples using stratified sampling based on the graph instance, aiming for balanced representation across all instances. For some graph instances, achieving perfect balance may not be possible due to the filtering process removing a significant portion of examples.

We finally take the number of recombination examples as shown in Table 4.9. We use 128 examples for each dataset to be used for validation.

4.8.2 Training Runs, Hyperparameters, and Curves

We train the diffusion recombination models for 50 epochs, with a batch size of 64 for ER-50-100 and ER-300-400, and 32 for ER-700-800. We use the same training hyperparameters we used for training Difusco in Section 4.3. This time, we use a single NVIDIA A100 80GB (SXM4) GPU for the ER-50-100 dataset, and four of the same

Table 4.9: Number of recombination examples used for training the diffusion recombination model across different datasets.

Dataset	$N_{\text{rec-train}}$
ER-50-100	159,459
ER-300-400	168,259
ER-700-800	143,425

GPUs for the ER-300-400 and ER-700-800 datasets using single-node distributed training. We show the training curves for the ER-700-800 dataset in Figure 4.6.

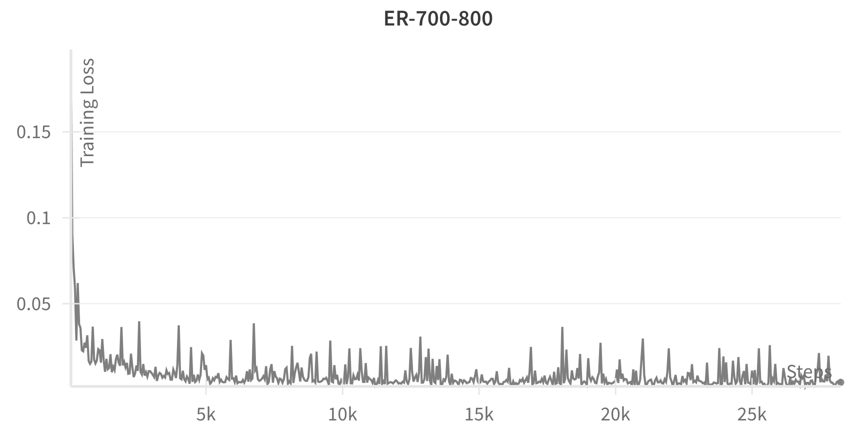

Figure 4.6: Training loss trajectory for the diffusion recombination model on the ER-700-800 dataset.

The training for the three models converges around 0.003 loss, which is lower than the loss of the Difusco models. This can be explained by the fact that the diffusion recombination model is trained with a larger dataset. $N_{\text{rec-train}}$ is around 4 times larger than the number of training examples used for training Difusco, which is 39,872.

4.9 Diffusion-based Recombination Evaluation

In this section, we systematically evaluate the performance of the diffusion recombination model in different settings.

4.9.1 Isolated Evaluation of the Diffusion Recombination Model

We start by evaluating the performance of the diffusion recombination model in isolation, without the rest of the EA framework. We compare the following methods:

- Diffusion Recombination: the trained diffusion recombination model.

- Difusco Inference: the basic Difusco model used for inference. In this case, the parents of the recombination example are not used.
- Classical Recombination: the classical recombination operator used in the baseline EA, as explained in Section 3.1.

For the Diffusion Recombination, we compare the conditioning on three different types of parent solutions:

- EA Parents: the parents of the recombination example are used as input to the diffusion recombination model. As explained in Section 4.8.1, the parents are part of a population initialized with Difusco sampling, which means the parents are high-quality solutions, likely better than greedy-heuristic solutions.
- Heuristic: the parents of the recombination example are generated using the decoding heuristic of the baseline EA using random node probabilities.
- Random Noise: the parents of the recombination are infeasible random noise vectors.

We show the results in Table 4.10. We measure the gap to the optimal recombination solution, averaged over the test datasets. For each pair of parents, we sample two offspring solutions and compute the gap to the optimal recombination label considering the best offspring. This is the same setting used in the diffusion-based recombination considered in the EA, as explained in Section 3.2.

Table 4.10: Performance comparison of different recombination approaches measured by the average gap to the optimal recombination label. Results are reported for the three datasets.

Method	Parent Type	Gap (%)		
		ER-50-100	ER-300-400	ER-700-800
Diffusion Recombination	EA Parents	**1.28**	**6.10**	**5.18**
	Heuristic	2.83	12.18	16.81
	Random Noise	3.10	26.28	26.60
Difusco Inference	–	6.36	8.51	5.44
Classical Recombination	EA Parents	6.38	20.52	24.13

The results for the Diffusion Recombination method align with our expectations: conditioning on higher quality parents results in improved performance. Specifically, we observe the following order of effectiveness: EA parents < Heuristic parents < Random Noise parents.

Moreover, the diffusion recombination model performs better than the Difusco Inference method on the three datasets, although not by a large margin in ER-700-800. This comes from the fact that the training labels were optimal recombination samples that improved the Difusco sampled population in the first place.

The previous results are promising, as they show that the diffusion recombination model (1) can approximate the optimal recombination operator to a high degree of accuracy, and (2) satisfies the property that higher quality parents lead to better offspring solutions.

4.9.2 Evaluating the Cost Monotonicity Property

Cost monotonicity is a desirable property of recombination operators in evolutionary algorithms. For a recombination operator $f(x, y)$ that combines parent solutions x and y to produce offspring, cost monotonicity refers to the relationship between the quality of the parent solutions and the resulting offspring. Formally, if $c(x)$ represents the cost (or quality) of solution x, a recombination operator $f(x, y)$ exhibits cost monotonicity when the cost of the offspring $c(f(x, y))$ correlates positively with the costs of the parents $c(x)$ and $c(y)$. This property is crucial for evolutionary algorithms as it ensures that improvements in the parent population propagate effectively to future generations.

To evaluate this property in our diffusion recombination model, we designed an experiment using 20 test instances from each dataset. For each instance, 20 feasible MIS solutions of increasing quality were generated using Gurobi, then paired into 10 parent pairs of increasing mean quality. The diffusion recombination operator was applied to these 10 parent pairs, sampling two solutions per pair following the same procedure used in the diffusion-based recombination. For each pair, the mean parent cost and the best cost among their offspring were recorded. This process was repeated across all 20 instances, with results shown in Figure 4.7. The figure presents density heatmaps for each dataset, complemented by linear regression lines to visualize the trend.

The results indicate that the diffusion recombination model exhibits moderate cost monotonicity across all datasets. While the linear regression p-value does not exhibit statistical significance (at the 0.05 level) for the ER-50-100 dataset, an increasing trend is observable across all datasets. Additionally, the higher density consistently visible in the left-bottom corner of the plots confirms that high-quality parents tend to produce high-quality offspring. The bi-clustered nature of the plot for ER-700-800 does not necessarily indicate a meaningful trend, as this behavior is likely to dissipate when analyzing a larger sample of graphs.

It is worth noting that in this experiment, the diffusion recombination model operates mostly on parent solutions that lie outside its training distribution. The model was primarily trained on high-quality parent-child pairs, which explains why the cost monotonicity relationship might not be as pronounced when evaluating across the full quality spectrum.

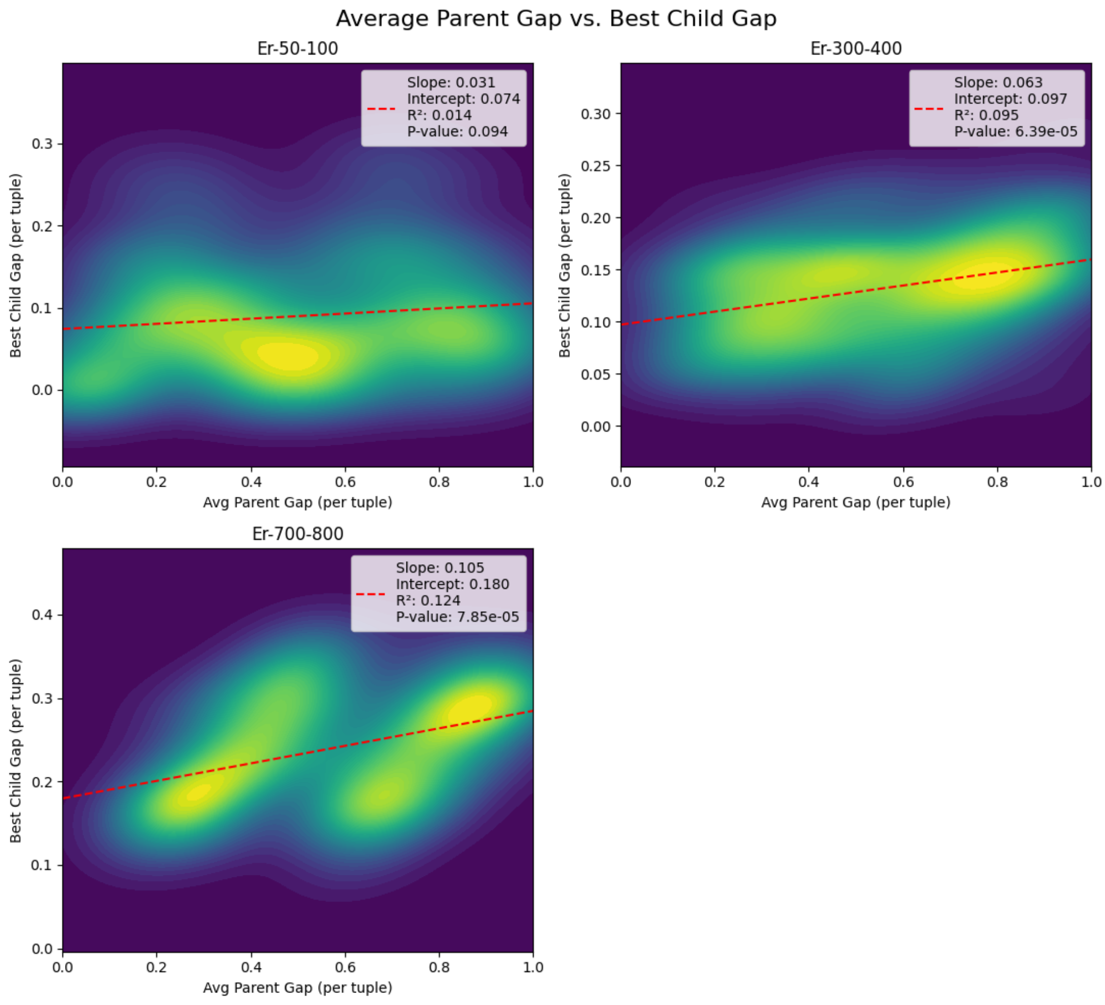

Figure 4.7: Cost monotonicity visualization through scatter heatmaps for the diffusion recombination model across all datasets.

4.9.3 Evaluation of the Diffusion Recombination within the EA

After proving the effectiveness of the diffusion recombination model in isolation, we now integrate it within the EA framework and evaluate its performance.

Table 4.11 compares the performance of the following five variants of the MIS EA, including the final DEA framework:

1. Naïve EA: we use classical recombination, greedy initialization, and $P = 50$. We set the number of generations in a way that the runtime is similar to the full DEA($P = 16, G = 20$) algorithm. This leads to $G = 50, 75, 305$ generations for ER-50-100, ER-300-400, and ER-700-800 respectively.

2. Naïve EA with Difusco initialization: since we sample from Difusco, we set $P = 16$. We set the same number of generations as the Naïve EA.

3. Optimal recombination: this is the expert EA that we would want to replicate. We use Difusco initialization, $P = 16$, and $G = 20$ for ER-50-100, $G = 5$ for ER-300-400, and $G = 3$ for ER-700-800.

4. Diffusion Recombination with greedy initialization: we set $P = 16$ and $G = 20$. Initialization is done using the greedy heuristic. This setting allows us to isolate the impact of Difusco initialization in the DEA framework.

5. Full DEA: Diffusion Recombination with Difusco initialization. We set $P = 16$ and $G = 20$.

We report the gap improvement with respect to the initial population gap (i.e. the 16-shot Difusco gap), the average gap to the optimal label, the initial Difusco gap, and the average runtime per instance and per iteration in seconds.

Table 4.11: Performance comparison of different EA variants across datasets: diffusion vs greedy initialization, and classical vs optimal vs diffusion recombination.

Dataset	Recomb.	Init	Gap Improv	Gap (%)	G	P	Runtime (s)	Runtime/G (s)
ER-50-100	Classic	DS	-1.53	0.55	50	16	24.1	0.5
		RG	-1.82	0.25	50	50	34.6	0.7
	Optimal	DS	-2.07	0.00	20	16	11.4	0.6
	Diffusion	DS	-2.01	0.07	20	16	12.7	0.6
		RG	-1.97	0.11	20	16	12.5	0.6
ER-300-400	Classic	DS	-0.54	3.57	75	16	98.6	1.0
		RG	4.47	8.58	75	16	95.5	0.6
	Optimal	DS	-3.53	0.58	5	16	441.4	88.3
	Diffusion	DS	-3.17	0.94	20	16	109.1	5.5
		RG	-2.73	1.39	20	16	109.0	5.4
ER-700-800	Classic	DS	-0.34	5.75	305	16	431.7	0.8
		RG	7.27	13.36	305	16	438.0	0.8
	Optimal	DS	-2.88	3.21	3	16	314.4	104.8
	Diffusion	DS	-3.28	2.81	20	16	475.1	23.8
		RG	-1.57	4.52	20	16	473.7	23.7

From the results, we get the following insights:

- The full DEA framework exhibits comparable performance to the expert EA. The differences in the final gaps are minimal (0.04 points on ER-50-100, 0.34 points on ER-300-400, -0.4 points on ER-700-800). This confirms that the diffusion recombination model is able to approximate the optimal recombination operator to a high degree of accuracy, and replacing the optimal operator with the learned operator does not result in a significant performance drop. The case of ER-700-800 is particularly interesting because we achieve an improvement of 0.4 points over the expert EA. This is possible because we run DEA for 20 generations instead of 3, which allows the diffusion recombination model to have more time to improve the population.
- The diffusion recombination operator performs well even when the population is initialized using the greedy heuristic. This suggests that the model is able to generalize to out-of-distribution instances since the training data was gathered during EA runs initialized with Difusco sampling. The performance of this version, however, is still worse than the full DEA framework. From this ablation, we conclude that the Difusco initialization is essential to the performance of the DEA framework.
- The full DEA framework outperforms both versions of the Naïve EA by a significant margin when given the same time limit constraints. When comparing to the version with Difusco initialization, which performs naturally better than the greedy initialization, the differences are still significant (0.38 points on ER-50-100, 2.63 points on ER-300-400, and 2.94 points on ER-700-800). This can be seen as the ablation study that confirms that the optimal recombination operator is essential to the performance of the DEA framework.
- The runtime of the full DEA framework is dominated by the diffusion inference, which happens every generation at the recombination step. As observed in Section 4.5, the runtime scaling with the population size is linear instead of sublinear and is high for large graphs and large population sizes (e.g. 8s for ER-300-400 and 43s for ER-700-800). This makes the full DEA framework computationally expensive. However, the runtime per iteration is much lower than the runtime of the optimal recombination operator. Moreover, given the fact that the performance is close to the optimal recombination operator, we can conclude that we successfully leveraged a learning-based approximation in order to speed up the recombination step.

4.10 Benchmarking DEA Against Classical and Learning-Based Solvers

We evaluate the full DEA algorithm against various state-of-the-art solvers, including the following:

- Difusco, sampling $P \in \{4, 8, 16, 32\}$ times in parallel and taking the best solution.

- The full DEA algorithm, using $P = 16, G = 20$, Difusco initialization, and diffusion recombination.
- T2T [LGWY23], sampling 1 or 4 times. This represents a state-of-the-art diffusion-based solver that combines supervised training and unsupervised gradient-based search. We do not train the model for ER-50-100 and ER-300-400, and we simply report the results of the T2T authors.
- KaMIS, using a time limit of 60 seconds.
- Gurobi 12.0 solving the formulation in 2.1.4. We set a time limit equal to the mean runtime of the DEA algorithm for each dataset (12.7s for ER-50-100, 109.1s for ER-300-400, 497.5s for ER-700-800). It operates with the parameter `MIPFocus=1` for focus on the primal side, and a limit of 4 threads.

Table 4.12 shows the average cost and runtime of each method across the three datasets.

Table 4.12: Comparison of average cost and runtime in seconds for different MIS solvers across datasets.

Method	ER-50-100		ER-300-400		ER-700-800	
	Cost	Time (s)	Cost	Time (s)	Cost	Time (s)
Difusco x4	19.87	0.6	34.28	2.5	40.74	11.1
Difusco x8	20.12	0.6	35.15	4.5	41.46	21.9
Difusco x16	20.32	0.7	35.53	8.7	42.12	43.6
Difusco x32	20.52	0.9	35.89	17.2	42.51	85.3
DEA (ours)	20.73	12.7	36.70	109.1	43.43	475.1
T2T x1	–	–	–	–	39.56	4.0
T2T x4	–	–	–	–	41.37	13.1
KaMIS	20.75	60.0	37.05	60.0	44.85	60.0
Gurobi	20.75	0.1	35.57	109.2	41.36	475.5

From the results, we can draw the following conclusions:

- The full DEA framework outperforms Difusco sampling, even when using 32-shot inference. Given the curves observed in Section 4.2, there is no hope that doubling the number of shots will improve the performance of Difusco enough to lead to results better than the DEA. Therefore, we conclude that the DEA is a valid extension of plain Difusco, enhancing performance beyond its Difusco's multi-shot capabilities.
- T2T x4 outperforms Difusco x4, but is worse than the other Difusco variants (x8, x16, x32). It is also significantly worse than the DEA algorithm. This suggests that the gradient-based search is effective, but yet cannot compete with heuristic-based

search methods to improve the sampled solutions, as we do in the DEA framework. If the T2T approach exhibits the multimodal property, we could consider a variant of the DEA that uses it for initialization or recombination, instead of the current Difusco-based components.

- The ER-50-100 dataset is solved to optimality by the classic solvers, and Gurobi gives a proof of optimality. In this dataset, the DEA algorithm achieves a gap of 0.07% from the optimal solution, requiring only 12.7 seconds.

- Notably, the DEA algorithm outperforms Gurobi on the large datasets marginally. Gurobi is a state-of-the-art MIP solver, but it is not specialized for any type of CO problem. It is therefore a good result to see that our DEA framework, which is in principle non-specialized, strongly outperforms it. While the 32-shot version of Difusco already matches the Gurobi results on the three datasets, the key result is that the DEA framework is able to marginally increase the improvements: DEA's best costs are 3.18% better than Gurobi on ER-300-400, and 5.01% better on ER-700-800, while Difusco is only 0.89% and 3.02% better, respectively.

- As many ML-based solvers, DEA struggles to compete with state-of-the-art specialized classical solvers. KaMIS exhibits superior performance on the large datasets, using only a time limit of 60 seconds. In the case of ER-300-400, the DEA demonstrates a gap of 0.94% compared to KaMIS, but it takes an average of 109.1 seconds. For ER-700-800, the DEA exhibits a gap of 2.81% relative to KaMIS, at the cost of an average runtime of 475.1 seconds.

4.11 Generalization to Out-of-Distribution Instances

In this section, we evaluate the generalization of Difusco and the DEA framework to out-of-distribution instances. We consider one dataset of 42 Erdős-Rényi graphs with graph sizes $1300 \leq n \leq 1500$, which are significantly larger than the ones used for training. They are generated with the same edge probability $p = 0.15$. We compare the performance of the following methods:

- Difusco trained on ER-50-100, ER-300-400, and ER-700-800, both for 1-shot and 16-shot.

- DEA ($P = 16, G = 20$), with its diffusion components using the checkpoints trained for ER-700-800.

- The ground truth optimal solution found by the KaMIS solver with a time limit of 60 seconds.

Table 4.13 reports the average cost and the total inference time for each method.

Table 4.13: Comparison of average cost and inference runtime per instance for different methods on ER-1300-1400.

Method	Cost	Runtime / Instance (s)
Difusco x1 (ER-50-100)	34.69	1.1
Difusco x1 (ER-300-400)	34.45	1.3
Difusco x1 (ER-700-800)	34.40	2.0
Difusco x16 (ER-50-100)	37.07	1.5
Difusco x16 (ER-300-400)	37.12	8.1
Difusco x16 (ER-700-800)	37.07	45.4
DEA ($P = 16, G = 20$, ER-700-800)	43.38	904.8
KaMIS	49.90	60

The results show that plain Difusco exhibits very poor generalization to out-of-distribution instances. The 16-shot inference only improves the 1-shot version by 7.8% on the ER-50-100 dataset, which suggests that when applied to out-of-distribution instances, the multimodal property is not preserved. Nevertheless, the bad performance is expected, as the models were trained on a significantly smaller dataset than the one used for testing. However, it would have been more natural to see the checkpoints trained on the largest dataset (ER-700-800) perform slightly better than the ones trained on the smaller datasets, but this is not what we observe.

The DEA framework, on the other hand, shows much better generalization to out-of-distribution instances. This is expected since we are essentially embedding Difusco within a population-based search framework. It is worth noting that the DEA average gap (13.1%) is still significantly higher and would naturally benefit from training on the 1300-1500 instances.

CHAPTER 5

Conclusions

This chapter concludes the thesis. In Section 5.1, we present the main findings and contributions of our work. Section 5.2 revisits the research questions posed at the beginning of the thesis and provides answers based on our experimental results. Section 5.3 discusses the limitations of our approach, while Section 5.4 outlines promising future research directions and extensions. Finally, Section 5.5 offers concluding thoughts on the significance and implications of this research.

5.1 Summary of Key Findings and Contributions

This thesis set out to improve ML-based combinatorial optimization by integrating denoising diffusion models with evolutionary algorithms. Our primary goal was to address the lack of a unified framework that integrates the generative power of diffusion models with the problem-independent exploratory strengths of EAs. Our main contribution is a novel hybrid framework, called DEA, that synergistically combines diffusion models with evolutionary algorithms.

The experimental results demonstrate the viability and efficacy of this hybrid approach on the NP-hard Maximum Independent Set (MIS) problem. We focus on three datasets of different sizes of Erdős-Rényi graphs, where the DEA framework achieved significant improvements over the standalone Difusco, which represents the state-of-the-art for diffusion-based CO. Through our experiments, we provide evidence that the multimodal property of diffusion models provides the diversity for evolutionary search, while evolutionary search mechanisms effectively overcome the limitations of the diffusion model.

Our work demonstrates that diffusion-based recombination operators can be successfully learned through imitation learning and can effectively combine solution features even outside their training distribution. The EA search, which includes this diffusion-based recombination operator, is able to improve the solutions of the initial population sampled by Difusco by a large margin (-3.17 points for ER-300-400 and -3.28 points for ER-700-800). Our empirical results also show that DEA significantly outperforms plain Difusco on out-of-distribution instances, confirming that population-based approaches enhance the resilience of generative models in combinatorial spaces.

Notably, the DEA framework marginally outperforms Gurobi under the same time limit on the large datasets, with improvements of 3.18% on ER-300-400 and 5.01% on ER-700-800. Despite these advancements, we also identify remaining challenges when compared to specialized classical solvers such as KaMIS, where DEA still exhibits gaps of 0.94% on ER-300-400 and 2.81% on ER-700-800, while requiring significantly more computation time. For the smallest dataset (ER-50-100), DEA achieves near-optimal solutions with a gap of only 0.07% from the proven optimal solutions.

5.2 Revisiting the Research Questions

At the beginning of this thesis, we outlined several research questions to steer our investigation. In this section, we will respond to each question based on the insights gained from our experimental findings:

Understanding Difusco's strengths and limitations

How much of the success of Difusco is due to the posterior local search heuristics?

We answer this question in Section 4.2 by studying the TSP post-processing heuristics employed in the work of [SY23]. The main findings are that post-processing heuristics are critical for the performance of the Difusco approach. Without them, Difusco is not better than a simple, general, and faster greedy approach. However, when paired with the 2-opt heuristic or MCTS, Difusco delivers very high-quality solutions that require advanced solvers to beat, reaching state-of-the-art performance compared to other neural solvers.

Is the multimodal property of Difusco observable?

We answer this question in Section 4.4. We were able to clearly observe the multimodal property of Difusco for both the TSP and MIS problems: more samples lead to a higher quality of the best solution in the population, although with diminishing returns. This property motivates the use of a population-based approach as the proposed DEA.

How do Difusco's runtimes scale with the problem size and the number of processed instances in parallel?

We address this question in Section 4.5. Our analysis of parallel Difusco inference yields underwhelming results. Only the ER-50-100 dataset shows a strong sublinear scaling, while the ER-300-400 and ER-700-800 datasets closely follow linear scaling. This indicates that the parallelization of the inference process is not efficient.

When it comes to the greedy decoding heuristic, we successfully vectorize the algorithm using PyTorch. The scaling experiments show a linear scaling with the duplication of batch size, which is the desired behavior.

Evaluating the DEA framework

How does the DEA compare to other solvers in terms of solution quality and computational time?

A first answer to this question is given in Section 4.9.3. We show that the full DEA framework outperforms the Naïve EA on the three MIS datasets in terms of solution quality by a significant margin, also when starting from the same initial population (either Difusco or Greedy). When it comes to the runtime, we observed that the DEA is not able to scale well to larger instances because the runtime of the DEA is highly dominated by the diffusion inference, which happens at every generation. For large graphs (e.g., ER-700-800), the implication is that the DEA takes 475s to process a single instance, and this long runtime poses as the main limitation of the DEA.

We complete the answer to this question in Section 4.10, where we compare the DEA to other solvers on the three datasets. Our experimental results show that the DEA framework significantly outperforms Difusco sampling, even with 32-shot inference, highlighting the necessity of our approach for improving Difusco's performance. Interestingly, DEA marginally outperforms Gurobi on larger datasets, with improvements of 3.18% on ER-300-400 and 5.01% on ER-700-800. However, like many ML-based solvers, DEA still struggles to compete with specialized classical solvers such as KaMIS, exhibiting gaps of 0.94% on ER-300-400 and 2.81% on ER-700-800, while requiring significantly more computation time. For the smallest dataset (ER-50-100), DEA achieves near-optimal solutions with a gap of only 0.07% from the proven optimal solutions.

How does the equivalent method without diffusion-based operators perform? What is the contribution of the diffusion components?

We answer this question in Section 4.9.3, where we perform an ablation study to understand the contribution of both the diffusion recombination operator and the diffusion initialization. The results show that both components are crucial for the performance of the DEA. In particular, we find that the recombination operator is more important than the initialization because the performance drops are more severe when removing it. This also suggests that the diffusion recombination can perform well out of distribution, as it is able to improve populations that are initialized greedily instead of using Difusco sampling, as in the training set.

How does the DEA perform in terms of generalization to out-of-distribution instances?

We answer this question in Section 4.11 by evaluating the generalization capabilities of both Difusco and our DEA framework on significantly larger Erdős-Rényi graphs. Our experiments used 42 graphs with sizes $1300 \leq n \leq 1500$, which are substantially larger than any instances in the training datasets. The results demonstrate that plain Difusco exhibits poor generalization to out-of-distribution instances, both with 1- and 16-shot inference. We find evidence that the multimodal property of Difusco is not well-preserved out-of-distribution.

In contrast, the DEA framework showed substantially better generalization capabilities, achieving an average cost of 43.38 compared to Difusco's best performance of 37.12. While the DEA still has a 13.1% gap from the ground truth solutions (found by KaMIS), it significantly outperforms all Difusco variants. This improved generalization capability is not a surprising result, as it is attributable to the population-based evolutionary approach that enhances exploration beyond what the base diffusion model can achieve.

5.3 Limitations

Despite the promising results of our DEA framework, several important limitations must be acknowledged:

- **Computational Expense:** The diffusion components of the DEA framework are computationally expensive and require GPU both for training and inference. This dominates the runtime of the algorithm and is the main limiting factor for the scalability of the DEA. Other solvers like KaMIS achieve superior results using only CPU resources.

- **Resource-Intensive Training:** The synthetic data generation scheme used to train the recombination operator is extremely resource-intensive, requiring substantial GPU hours that may be prohibitive in many research settings. This is because the training process requires running the EA with Difusco-based initialization (requiring a GPU) and optimal recombination (dominating the runtime).

- **Inefficient Parallelization:** Our current implementation fails to leverage batched parallel inference with the expected sublinear scaling, limiting efficiency gains for larger problem instances.

- **Problem-Specific Components:** While the overall DEA framework is problem-independent in the sense that its success is not due to problem-specific components, training the recombination operator remains problem-dependent, as it requires finding an optimal recombination strategy specific to each combinatorial problem that acts as a demonstrator for the diffusion recombination operator.

- **Limited Out-of-Distribution Generalization:** Although DEA demonstrates improved out-of-distribution generalization compared to plain Difusco, it still exhibits significant performance degradation when operating on larger instances beyond the training distribution. This highlights that out-of-distribution generalization in ML-based combinatorial optimization remains an open challenge.

5.4 Future Directions and Extensions

These limitations highlight important areas for future research and development. Building on our findings, we identify several promising directions:

Technical Extensions

- **Expanded Problem Domains:** Benchmark the DEA framework on other combinatorial optimization problems beyond MIS. This would validate the generality of our approach and potentially identify problem domains where the synergy between diffusion models and EAs is particularly effective.
- **Parallelization Optimization:** Develop improved parallelization strategies to leverage the theoretically possible sublinear scaling in diffusion inference. This could significantly reduce computational costs, especially for larger problem instances, making the DEA framework more competitive in terms of runtime.
- **Efficient Data Generation:** Explore alternative synthetic data generation schemes for training the diffusion recombination operator that require fewer computational resources. This could include using lower quality non-diffusion initializations for the synthetic EA trajectories or developing more efficient demonstrators for the recombination process.
- **Diffusion-Based Mutation:** Develop a diffusion-based mutation operator to complement the recombination operator. This could be trained separately or could reuse the recombination model with identical parents, though some performance degradation might be expected with the latter approach.
- **Advanced Diffusion Backbones:** Incorporate recent improvements to Difusco (e.g., T2T [LGWY23], DISCO [YZH+24]) as the backbone for the diffusion operators in DEA. These more advanced models could provide better initialization and recombination, potentially improving the overall performance of the framework.

Broader Research Directions

- **Other ML-Metaheuristic hybrids:** Explore hybridizing other metaheuristics beyond EAs with diffusion models. This could include approaches like simulated annealing, tabu search, or ant colony optimization, each potentially offering different complementary strengths to the generative capabilities of diffusion models.
- **Adaptive Operator Selection:** Develop mechanisms for adaptive selection between different operators (traditional EA operators vs. diffusion-based ones) based on their effectiveness during different stages of the search process. Given the computational expense of the diffusion components, it is reasonable to develop mechanisms to reduce the number of inference calls to the diffusion model.

- **Theoretical Foundations:** Establish stronger theoretical foundations for understanding why and how diffusion models enhance evolutionary search. This could involve analyzing the diversity and quality of solutions generated by diffusion models and their impact on the convergence properties of EAs.

- **Multi-Objective Optimization:** Extend the DEA framework to handle multi-objective combinatorial optimization problems, leveraging the multimodal property of diffusion models to explore diverse regions of the Pareto front.

The fusion of diffusion models with evolutionary algorithms opens a promising new direction in combinatorial optimization. As both fields continue to advance, we anticipate that frameworks like DEA will play an increasingly important role in solving complex combinatorial problems, potentially bridging the gap between specialized classical solvers and learning-based approaches.

5.5 Final Remarks

In this thesis, we have demonstrated that combining the generative capabilities of diffusion models with the exploratory mechanisms of EAs creates a powerful framework for solving CO problems. The DEA framework represents a significant step forward in ML-based combinatorial optimization, bridging the gap between generative approaches and traditional metaheuristics. While challenges remain—particularly in computational efficiency—our results clearly show that this hybrid approach outperforms its constituent parts. As machine learning continues to advance and computational resources become more affordable, frameworks like DEA promise to become increasingly valuable tools in the combinatorial optimization toolbox. The integration of learning-based components with search-based methods represents a promising direction not just for the specific problems studied here, but for the broader field of computational intelligence. We believe this work contributes to a growing understanding that the future of optimization lies not in choosing between classical algorithms and machine learning approaches, but in thoughtfully combining their complementary strengths.

Appendix

Difusco Hyperparameters

Maximum Independent Set We set the architecture of the GNN to have $d_h = 256$, which is the embedding dimension, and $n_l = 12$, which is the number of layers. According to the findings of [SY23], SATLIB instances are better modeled with categorical diffusion, while ER graphs are better modeled with Gaussian diffusion, and we follow this recommendation.

For inference, we use a cosine inference schedule. We set the number of inference denoising steps to $T_{inf} = 50$, which is a critical hyperparameter that trades off solution quality and inference runtime.

For training, we use a linear diffusion schedule with $T_{train} = 1000$ steps, and again a cosine inference schedule. We use a batch size of $b = 64$, a learning rate of 0.0002, a weight decay of 0.0001, and a learning rate scheduler set to "cosine-decay".

Traveling Salesperson Problem We set the architecture of the GNN to have $d_h = 256$, which is the embedding dimension, and $n_l = 12$, which is the number of layers. The type of diffusion is set to categorical. For inference, we use a cosine inference schedule with $T_{inf} = 50$ steps.

Graph Neural Networks

Graph Neural Networks (GNNs) have emerged as a powerful framework for learning representations of graph-structured data, enabling breakthroughs in social network analysis [KW16], molecular property prediction [DMAI$^+$15], and recommendation systems [YHC$^+$18]. Unlike traditional deep learning models for grid-like data (e.g., CNNs for images), GNNs explicitly model relational inductive biases through message passing between nodes, capturing both local structure and global topology. GNNs excel in handling irregular, non-Euclidean data, making them essential for tasks like protein interaction prediction [FBSBH17] and financial fraud detection [LCY$^+$21].

Fundamentals of message passing At their core, GNNs iteratively refine node representations by aggregating information from neighborhoods. For a graph $G = (V, E)$

with node features $\mathbf{h}_v^{(0)} \in \mathbb{R}^d$ initialized from input data, the k-th layer updates node v's representation via:

$$\mathbf{h}_v^{(k)} = \phi^{(k)}\left(\mathbf{h}_v^{(k-1)}, \bigoplus_{u \in \mathcal{N}(v)} \psi^{(k)}(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)})\right),$$

where $\psi^{(k)}$ is the message function encoding interactions between node v and neighbor u, $\bigoplus$ represents permutation-invariant aggregation (e.g., sum, mean, max), and $\phi^{(k)}$ is the update function that combines aggregated messages with v's previous state.

Key architectural variants include:

- **Graph Convolutional Networks (GCN)** [KW16]: Aggregates normalized neighbor features via:

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}^{(k)} \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{\mathbf{h}_u^{(k-1)}}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}}\right)$$

- **Graph Attention Networks (GAT)** [VCC+18]: Learns attention weights α_{vu} for neighbor importance:

$$\mathbf{h}_v^{(k)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu} \mathbf{W}^{(k)} \mathbf{h}_u^{(k-1)}\right)$$

- **Anisotropic Graph Neural Network** [BL18]: Introduces anisotropic diffusion through learned edge-wise aggregation weights:

$$\mathbf{h}_v^{(k)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \mathbf{\Theta}_{vu}^{(k)} \mathbf{h}_u^{(k-1)}\right)$$

  where $\mathbf{\Theta}_{vu}^{(k)}$ are learnable parameters for each edge (v, u).

Training paradigms GNNs are typically trained end-to-end with task-specific objectives:

- **Node Classification**: Cross-entropy loss over labeled nodes:

$$\mathcal{L} = -\sum_{v \in \mathcal{V}_{\text{lab}}} y_v \log \sigma(\mathbf{W}_c \mathbf{h}_v^{(K)})$$

  where $\mathcal{V}_{\text{lab}}$ are labeled nodes and $\mathbf{W}_c$ maps final-layer embeddings to classes.

- **Link Prediction**: Score edges via decoder $f_{\text{dec}}(\mathbf{h}_u^{(K)}, \mathbf{h}_v^{(K)})$ (e.g., dot product) and optimize pairwise BCE loss:

$$\mathcal{L} = -\sum_{(u,v) \in \mathcal{E}_{\text{pos}}} \log f_{\text{dec}}(u, v) - \sum_{(u,v) \in \mathcal{E}_{\text{neg}}} \log(1 - f_{\text{dec}}(u, v))$$

Neural Algorithmic Reasoning In neural algorithmic reasoning [VB21], which aims to replicate classical algorithmic behaviors with neural networks, GNNs have become a pivotal tool due to their alignment with iterative, rule-based processes. For example, GNNs naturally mirror the logic of algorithms like shortest path computation or graph traversal, where iterative updates and neighborhood interactions are central. Though neural algorithmic reasoning offers promising synergies between learning and classical computation, applying it to CO problems is still an open challenge [GNBL23].

Overview of Generative AI Tools Used

In this document, several generative AI models and tools have been employed as aids throughout the research and writing process. These tools were utilized to enhance the clarity, structure, and organization of my work, but not to replace my own intellectual contribution.

The following generative AI models and tools were used:

- Chat-based models:
  - OpenAI models: GPT-4o, GPT-4o-mini, and OpenAI o3-mini.
  - Anthropic models: Claude 3.5, Claude 3.7, and Claude 3.7-thinking. I used these models within the Cursor IDE.
  - DeepSeek R1 and DeepSeek R1-DeepThink.
  - Gemini 2.0 Pro and Gemini 2.0 Flash Thinking.
- Code generation models:
  - Cursor autocomplete, powered by a custom model by Anysphere Inc.
  - Cursor agent, powered by Claude 3.7.
- Translation and language proofing tools:
  - DeepL Translate
  - LanguageTool

These tools were used for the following purposes:

- **Structuring Text:** Assisting in developing schemas and outlines for different sections and chapters of the thesis. The AI models were prompted with my existing thesis content and external references to generate suggestions for the structure of sections and chapters.

- **Draft Generation:** Creating initial drafts of text passages, which were always subsequently reviewed, revised, and significantly improved by me. The AI models were prompted with my existing thesis content and external references to generate these starting points.
- **Idea Organization and Refinement:** Helping to organize and structure complex ideas, ensuring logical flow and coherence within the text. This included brainstorming and exploring different ways to present information effectively.
- **Information Retrieval:** In some instances, the search tool functionality available within certain AI interfaces was utilized to gather supplementary information related to specific topics, which then informed my writing process.
- **Language Proofing and Translation:** AI-powered tools such as DeepL Translate and LanguageTool were utilized for language proofing, including the translation of the abstract from English to German, as well as for checking the grammar and spelling of the final version of the thesis.

It is crucial to emphasize that while these tools provided valuable assistance, **absolutely no piece of writing in this thesis has been adopted without substantial modification and refinement by me.** The AI-generated content served solely as a starting point or suggestion, and the final text represents my own intellectual contribution and articulation of the research. The use of AI has been limited to enhancing my workflow and exploring different perspectives, while the core ideas, analysis, and conclusions are entirely my own.

List of Figures

List of Tables

List of Algorithms

Bibliography

[ABCC06] David Applegate, Robert Bixby, Vasek Chvatal, and William Cook. Concorde traveling salesman problem solver, 2006. Available at: `https://www.math.uwaterloo.ca/tsp/concorde/index.html`, Accessed: 23-Mar-2025.

[ABCC07] David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2007.

[AL03] E.H.L. Aarts and J.K. Lenstra. *Local Search in Combinatorial Optimization.* Princeton University Press, 2003.

[BFM97] Thomas Back, David Fogel, and Zbigniew Michalewicz. *Handbook of Evolutionary Computation.* Oxford University Press, New York, NY, 1997.

[BK94] Thomas Bäck and Sami Khuri. An evolutionary heuristic for the maximum independent set problem. *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 531–535 vol.2, 1994.

[BKK$^+$16] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.

[BKT$^+$22] Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[BL18] Xavier Bresson and Thomas Laurent. An experimental study of neural networks for variable graphs. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[BLMBT18] Radu Baltean-Lugojan, Ruth Misener, Pierre Bonami, and Andrea Tramontani. Strong sparse cut selection via trained neural nets for quadratic semidefinite outer-approximations. Technical report, Imperial College, London, 2018.

[BLP21] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[BMV+22] Zalán Borsos, Raphaël Marinier, Damien Vincent, Eugene Kharitonov, Olivier Pietquin, Matt Sharifi, Olivier Teboul, David Grangier, Marco Tagliasacchi, and Neil Zeghidour. Audiolm: a language modeling approach to audio generation. *IEEE/ACM Transactions on Audio Speech and Language Processing*, 2022.

[BPL+17] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[BR16] Christian Blum and Günther R. Raidl. *Hybrid Metaheuristics: Powerful Tools for Optimization.* Artificial Intelligence: Foundations, Theory, and Algorithms (AIFTA). Springer, 2016.

[BT97] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.

[CGJ96] Ed Coffman, M.R. Garey, and David Johnson. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing, 01 1996.

[Cha24] Stanley H. Chan. Tutorial on diffusion models for imaging and vision. *arXiv*, 2024. Available at: https://arxiv.org/abs/2403.18103, Accessed: 2025-03-23.

[CKG+21] Antonia Chmiela, Elias Boutros Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to schedule heuristics in branch and bound. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24235–24246, 2021.

[Cra97] Yves Crama. Combinatorial optimization models for production scheduling in automated manufacturing systems. *European Journal of Operational Research*, 99(1):136–153, 1997.

[CZZ+21] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J. Weiss, Mohammad Norouzi, and William Chan. Wavegrad: Estimating gradients for waveform generation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[DG97] M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[DJL$^+$22] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 23:1–48, 2022.

[DMAI$^+$15] David K Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy D Hirzel, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS'15: Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*, pages 2224–2232, 2015.

[DS04] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[ES15] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2015.

[Fal19] William Falcon. Pytorch lightning, 2019. Available at: `https://github.com/PyTorchLightning/pytorch-lightning`, Accessed: 23-Mar-2025.

[FBSBH17] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6533–6542, 2017.

[FL03] Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1):23–47, 09 2003.

[FL19] Felix Fey and Jan E. Lenssen. Fast graph representation learning with pytorch geometric. In *Proceedings of the ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[Fog62] Lawrence J. Fogel. Toward inductive inference automata. In *Proceedings of the International Federation for Information Processing Congress*, pages 395–399, 1962.

[FOW66] Lawrence J. Fogel, Alan J. Owens, and Michael J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.

[FSY24] Shengyu Feng, Zhiqing Sun, and Yiming Yang. DIFUSCO-LNS: Diffusion-guided large neighbourhood search for integer linear programming, 2024. Available at: `https://openreview.net/forum?id=9QV7Q9gKl9`, Accessed: 23-Mar-2025.

[GCF$^+$19] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences).* W. H. Freeman, 1979.

[Glo89] Fred Glover. Tabu search. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[GNBL23] Dobrik Georgiev, Danilo Numeroso, Davide Bacciu, and Pietro Liò. Neural algorithmic reasoning for combinatorial optimisation. In *Proceedings of the Second Learning on Graphs Conference (LoG 2023)*, PMLR 231, 2023.

[GO25] LLC Gurobi Optimization. Gurobi optimizer, 2025. Available at: `https://www.gurobi.com/`, Accessed: 23-Mar-2025.

[GWL$^+$25] Zijie Geng, Jie Wang, Xijun Li, Fangzhou Zhu, Jianye Hao, Bin Li, and Feng Wu. Differentiable integer linear programming. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

[Hel00] Keld Helsgaun. An effective implementation of the lin- kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 6840–6851, 2020.

[HKJ$^+$25] Seong-Hyun Hong, Hyun-Sung Kim, Zian Jang, Deunsol Yoon, Hyungseok Song, and Byung-Jun Lee. Unsupervised training of diffusion models for feasible solution generation in neural combinatorial optimization. *arXiv*, 2025. Available at: https://arxiv.org/abs/2411.00003, Accessed: 23-Mar-2025.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[Hoo12] Holger H. Hoos. Automated algorithm configuration and parameter tuning. In *Autonomous Search*, pages 37–71. Springer, 2012.

[HS00] Holger H. Hoos and Thomas Stützle. Satlib: An online resource for research on sat. In I.P. Gent, H. van Maaren, and T. Walsh, editors, *SAT 2000*, pages 283–292. IOS Press, 2000. Available at: `www.satlib.org`, Accessed: 23-Mar-2025.

[HS22] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2022.

[HSVW22] Emiel Hoogeboom, Víctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *Proceedings of the 39th International Conference on Machine Learning, PMLR*, volume 162, pages 8867–8887, 2022.

[HSY23] Junwei Huang, Zhiqing Sun, and Yiming Yang. Accelerating diffusion-based combinatorial optimization solvers by progressive distillation. In *Proceedings of the ICML 2023 Workshop: Sampling and Optimization in Discrete Space (SODS 2023)*, 2023.

[HWL$^+$21] Zeren Huang, Kerong Wang, Furui Liu, Hui-Ling Zhen, Weinan Zhang, Mingxuan Yuan, Jianye Hao, Yong Yu, and Jun Wang. Learning to select cuts for efficient mixed-integer programming. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[IL20] Takahiro Ishihara and Steffen Limmer. Optimizing the hyperparameters of a mixed integer linear programming solver to speed up electric vehicle charging control. In *Applications of Evolutionary Computation*, pages 37–53, Cham, 2020. Springer International Publishing.

[KAACA15] Heba Kurdi, Abeer Al-Anazi, Carlene Campbell, and Auhood Al Faries. A combinatorial optimization algorithm for multiple cloud service composition. *Computers and Electrical Engineering*, 42:107–113, 2015.

[Kar72] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[KBS$^+$16] Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 724–731. AAAI Press, 2016.

[KDM$^+$24] Lingkai Kong, Yuanqi Du, Wenhao Mu, Kirill Neklyudov, Valentin De Bortoli, Dongxia Wu, Haorui Wang, Aaron Ferber, Yi-An Ma, Carla P. Gomes, and Chao Zhang. Diffusion models as constrained samplers for optimization with unknown constraints. *arXiv*, 2024. Available at: https://arxiv.org/abs/2402.18012, Accessed: 23-Mar-2025.

[KDZ$^+$17] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, New Series*, 220(4598):671–680, 1983.

[KL20] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[KLKea24] R. Kemminer, J. Lange, J.P. Kempkes, and et al. Configuring mixed-integer programming solvers for large-scale instances. *Operations Research Forum*, 5:48, 2024.

[KLP17] Max Kruber, Marco E. Lübbecke, and Antoine Parmentier. Learning when to use a decomposition. In *Integration of AI and OR Techniques in Constraint Programming*, pages 202–210. Springer, 2017.

[KSPH21] Diederik P. Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 21696–21707, 2021.

[KV12] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms.* Algorithms and Combinatorics. Springer Berlin Heidelberg, 2012.

[KvHW19] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[KW16] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[LCY$^+$21] Zhiwei Liu, Chao Chen, Xuan Yang, Jun Zhou, Xiaoming Li, and Le Song. Pick and choose: A gnn-based imbalanced learning approach for fraud detection. *Proceedings of the Web Conference 2021*, pages 3168–3177, 2021.

[LD60] Ailsa H. Land and Alison G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[LDDB24] Anjian Li, Zihan Ding, Adji Bousso Dieng, and Ryne Beeson. Constraint-aware diffusion models for trajectory optimization. *arXiv*, 2024. Available at: https://arxiv.org/abs/2406.00990, Accessed: 23-Mar-2025.

[LGWY23] Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. From distribution learning in training to gradient search in testing for combinatorial optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[LH17] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[LLW$^+$24] Connor Lawless, Yingxi Li, Anders Wikum, Madeleine Udell, and Ellen Vitercik. Llms for cold-start cutting plane separator configuration. *arXiv*, 2024. Available at: https://arxiv.org/abs/2412.12038, Accessed: 23-Mar-2025.

[LMMV10] Andrea Lodi, Silvano Martello, Michele Monaci, and Daniele Vigo. Two-dimensional bin packing problems. *Paradigms of Combinatorial Optimization: Problems and New Approaches*, 2:107–129, 01 2010.

[LS23] B. Luteberget and G. Sartor. Feasibility jump: an lp-free lagrangian mip heuristic. *Math. Prog. Comp.*, 15:365–388, 2023.

[LSS96] Xingzhao Liu, Akio Sakamoto, and T. Shimamoto. A genetic algorithm for maximum independent set problems. *1996 IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems (Cat. No.96CH35929)*, 3:1916–1921 vol.3, 1996.

[LSS$^+$17] Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *J. Heuristics*, 23(4):207–229, 2017.

[LYC$^+$24] Ruihuai Liang, Bo Yang, Pengyu Chen, Xuelin Cao, Zhiwen Yu, Mérouane Debbah, Dusit Niyato, H. Vincent Poor, and Chau Yuen. Gdsg: Graph diffusion-based solution generator for optimization problems in mec networks. *arXiv*, 2024. Available at: https://arxiv.org/abs/2412.08296, Accessed: 23-Mar-2025.

[LYY$^+$25a] Ruihuai Liang, Bo Yang, Zhiwen Yu, Bin Guo, Xuelin Cao, Mérouane Debbah, H. Vincent Poor, and Chau Yuen. Diffsg: A generative solver for network optimization with diffusion model. *arXiv*, 2025. Available at: https://arxiv.org/abs/2408.06701, Accessed: 23-Mar-2025.

[LYY$^+$25b] Ruihuai Liang, Bo Yang, Chau Yuen, et al. Diffusion models as network optimizers: Explorations and analysis. *IEEE Internet of Things Journal*, 2025.

[LZL$^+$25] Haoyu Lei, Kaiwen Zhou, Yinchuan Li, Zhitang Chen, and Farzan Farnia. Boosting generalization in diffusion-based neural combinatorial solver via energy-guided sampling. *arXiv*, 2025. Available at: https://arxiv.org/abs/2502.12188, Accessed: 23-Mar-2025.

[MOS15] Renata Mansini, Wlodzimierz Ogryczak, and Maria Grazia Speranza. *Linear and Mixed Integer Programming for Portfolio Optimization.* Springer, 2015.

[ND21] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *Proceedings of Machine Learning Research*, volume 139, pages 8162–8171, 2021.

[PGC$^+$19] Adam Paszke, Sam Gross, Gustavo Chai, Edward Yang, Zachary Lin, Alban Desmaison, Andreas Kolesnikov, Jin Yu, Daniel Zhirnov, and Trevor C. Black. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 8026–8037, 2019.

[PR10] Sandro Pirkwieser and Günther R. Raidl. Variable neighborhood search coupled with ILP-based very large neighborhood searches for the (periodic) location-routing problem. In *Hybrid Metaheuristics*, pages 174–189, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[RBL$^+$22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10674–10685, 2022.

[RDN$^+$22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *arXiv*, 2022. Available at: https://arxiv.org/abs/2204.06125, Accessed: 23-Mar-2025.

[Rec73] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, 1973.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, 2015.

[RP06] Mauricio G. C. Resende and Panos M. Pardalos. *Handbook of Optimization in Telecommunications.* Springer, 2006.

[RvBW06] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming.* Elsevier, 2006.

[SALYS24] Lara Scavuzzo, Karen Aardal, Andrea Lodi, and Neil Yorke-Smith. Machine learning augmented branch and bound for mixed integer linear programming. *Mathematical Programming*, 2024.

[SBK22] Martin J. A. Schuetz, J. Kyle Brubaker, and Helmut G. Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4:367–377, 2022.

[SDWMG15] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of Machine Learning Research*, volume 37, 2015.

[SHL24] Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 43346–43367, 2024.

[SSDK$^+$21] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[SSHR24] Joan Salvà Soler, Vera C. Hemmelmayr, and Günther R. Raidl. Exact methods for the selective assessment routing problem. *Central European Journal of Operations Research*, 32(4):1–25, 2024.

[SY23] Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 3706–3731, 2023.

[TAM$^+$23] Nihat Engin Toklu, Timothy Atkinson, Vojtvech Micka, Pawel Liskowski, and Rupesh Kumar Srivastava. EvoTorch: Scalable evolutionary computation in Python. *arXiv*, 2023. Available at: `https://arxiv.org/abs/2302.12600`, Accessed: 23-Mar-2025.

[TV02] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematics, 2002.

[VB21] Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7):100273, 2021.

[VCC$^+$18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[VFJ15] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28, pages 2692–2700. Curran Associates, Inc., 2015.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[WCPZ22] Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors. *Graph Neural Networks: Foundations, Frontiers, and Applications.* Springer, 2022.

[Wol20] Laurence A. Wolsey. *Integer Programming.* Wiley, 2020.

[WWW$^+$24] Xuan Wu, Di Wang, Lijie Wen, Yubin Xiao, Chunguo Wu, Yuesong Wu, Chaoyu Yu, Douglas L. Maskell, and You Zhou. Neural combinatorial optimization algorithms for solving vehicle routing problems: A comprehensive survey with perspectives. *arXiv*, 2024. Available at: https://arxiv.org/abs/2406.00415, Accessed: 23-Mar-2025.

[WWY$^+$22] Haoyu Peter Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 31444–31454, 2022.

[YHC$^+$18] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, New York, NY, USA, 2018. Association for Computing Machinery.

[YWC$^+$24] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[YZH$^+$24] Kexiong Yu, Hang Zhao, Yuhang Huang, Renjiao Yi, Kai Xu, and Chenyang Zhu. Disco: Efficient diffusion solver for large-scale combinatorial optimization problems. *arXiv*, 2024. Available at: https://arxiv.org/abs/2406.19705, Accessed: 23-Mar-2025.

[ZHZ$^+$21] Jiongzhi Zheng, Kun He, Jianrong Zhou, Yan Jin, and Chu-Min Li. Combining reinforcement learning with lin-kernighan-helsgaun algorithm for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7386–7394, 2021.

[ZSC$^+$20] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 2712–2723, 2020.