Constraint-Aware Decoding in Graph-Based Diffusion Models for Combinatorial Optimization

Dissertation

COMP702 – M.Sc. project (2024/25)

Submitted by
Mohammed Faisal (201813631)

under the supervision of Supervisor Vladimir Gusev

Department of Computer Science
University of Liverpool

Student Declaration

I confirm that I have read and understood the University's Academic Integrity Policy.
I confirm that I have acted honestly, ethically, and professionally in conduct leading to assessment for the programme of study.
I confirm that I have not copied material from another source, committed plagiarism, nor fabricated data when completing the attached piece of work. I confirm that I have not previously presented the work or part thereof for assessment for another University of Liverpool module.
I confirm that I have not copied material from another source, nor colluded with any other student in the preparation and production of this work.
I confirm that I have not incorporated into this assignment material that has been submitted by me or any other person in support of a successful application for a degree of this or any other university or degree-awarding body.
SIGNATURE: Mohammed Faisal
DATE: September 25, 2025

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Dr. Vladmir Gusev, for his guidance, support, and valuable feedback throughout this research. I also thank the faculty and staff of the University of Liverpool for providing the resources and environment necessary for completing this dissertation. Finally, I am grateful to my parents for their encouragement and support during this journey.

Abstract

The Travelling Salesman Problem (TSP) is a well-known optimization challenge that involves finding the shortest possible route to visit a set of cities exactly once and return to the starting point [3, 4]. While classical algorithms, such as Christofides' heuristic [2, 26], provide near-optimal solutions with mathematical guarantees, recent advances in machine learning have opened up new possibilities for solving such problems [5, 6].

In this project, we explored constraint-aware decoding in graph-based diffusion models using the DIFUSCO framework [1]. Diffusion models, originally popular in image and text generation [7, 8], are adapted here to produce candidate solutions for combinatorial optimization tasks like TSP [9, 10]. However, these solutions often violate problem constraints [13, 14]. To address this, we integrated Christofides' algorithm into the decoding process to enforce valid tours and improve solution quality [46].

The experiments focused on the TSP50 dataset derived from Concorde benchmarks [16, 28]. We trained and evaluated DIFUSCO models, applied constraint-aware decoding, and compared the generated solutions against classical heuristics. Results showed that combining diffusion-based learning with Christofides' constraint enforcement improved both the validity and efficiency of solutions.

Statement of Ethical Compliance

This project was carried out in full compliance with the ethical guidelines of the University of Liverpool. The system developed is a deep learning-based chart pattern recognition tool for financial data. At no point did the work involve collecting, processing, or analysing sensitive personal data, and there was no need to recruit human participants for testing.

All data used in this project consisted of publicly available TSP Libraries and Dataset. Additionally, publicly available datasets for algorithm evaluation, such as the TSP dataset from `https://github.com/Spider-scnu/TSP`, were used. The use of public datasets raises no privacy concerns, and the outcomes of this work are intended solely for educational and decision-support purposes.

The technologies and libraries used, including Python, pandas, NumPy, Matplotlib, networkx, PyTorch, scikit-learn, and PyTorch, are all open-source or publicly available. Their use complied with proper licensing conditions and standard ethical software development practices.

Based on the University's framework, this project falls under Category B0, which means: **B0: No vulnerable or sensitive participant groups were involved.**

Furthermore, the project aligns with the BA0 considerations:

- **BA0: Data Sources** — Only publicly available datasets, such as the TSP dataset from `https://github.com/Spider-scnu/TSP`, and synthetic data generated programmatically were used.
- **BA0: Requirements Analysis and Evaluation** — All analysis and evaluation were performed using these datasets in compliance with standard academic and professional guidelines.

For these reasons, the project does not pose any ethical risks to individuals or groups. All design, implementation, and evaluation were conducted responsibly and in line with professional and academic standards.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivation

Combinatorial Optimization deals with finding the best solution from a finite set of possible solutions [3, 26]. It is a subfield of mathematical optimization. Some of the most common combinatorial optimization problems are the Travelling Salesman Problem (TSP) [28, 4], Knapsack Problem, and Minimum Spanning Tree (MST) [21, 22]. These problems are seen in daily life in the form of routing, logistics, network design, etc., where efficient decision-making is critical [5, 6]. Among these problems, the Travelling Salesman Problem (TSP) is a frequent occurrence due to its wide range of applications and simplicity in formulation [11, 7]. The goal of TSP is to start at a city, visit all other cities exactly once, and return to the starting point, finding the shortest possible tour by minimizing the distance or cost. The problem statement is classified as NP-Hard [3], which means that no polynomial-time algorithm is known to solve all instances optimally.

There are three categories of approaches to TSP:

1. **Exact solvers:** such as Concorde and Gurobi [16, 18], which guarantee optimal solutions but do not scale well to large instances due to exponential complexity.

2. **Heuristics and metaheuristics:** including 2-opt [36], Simulated Annealing, and the Lin–Kernighan–Helsgaun (LKH) algorithm [18], which are scalable but risk convergence to suboptimal solutions.

3. **Learning-based approaches:** such as Pointer Networks, Attention Models, NeuroCO, and AM-GCN [32, 30, 33, 44], which leverage Graph Neural Networks (GNNs) and Reinforcement Learning to learn data-driven heuristics or generative policies.

While learning-based methods demonstrate adaptability and generalisation, they often overlook hard constraints (e.g., visiting each node exactly once), which leads to infeasible or post-processed solutions [13, 14].

This project focuses on DIFUSCO (Graph-based Diffusion Solvers for Combinatorial Optimization) [1], which models combinatorial optimization as a generative denoising diffusion process over graphs. DIFUSCO iteratively denoises noisy graph states into high-quality solutions and has shown strong performance on benchmark problems like Max-Cut and TSP, outperforming several GNN- and RL-based baselines [9, 10]. However, DIFUSCO does not explicitly enforce feasibility during decoding, which can result in invalid solutions for tasks like TSP [13].

To address this limitation, we propose a hybrid approach combining DIFUSCO with Christofides' algorithm [46, 26]. Christofides' method, a classical approximation algorithm with a worst-case performance guarantee of 1.5, provides a principled way to construct feasible tours. By integrating Christofides either as a post-processing repair step or as a guided decoding mechanism within DIFUSCO, this project aims to unite the generative strengths of diffusion models with the theoretical guarantees of approximation algorithms. The outcome is a solver that is both probabilistically powerful and structurally reliable, producing feasible solutions with improved quality across synthetic and real-world datasets [17, 28].

1.2 Problem Statement

Recent advances in learning-based methods, particularly Graph Neural Networks (GNNs) [32, 30], Reinforcement Learning (RL) [42, 43], and diffusion models [7, 8], have shown promise in learning heuristics directly from data. DIFUSCO (Graph-based Diffusion Solvers for Combinatorial Optimization) [1] is one such framework that formulates combinatorial optimization as a generative denoising diffusion process over graphs, achieving competitive results on problems

like Max-Cut and TSP [9, 10]. However, DIFUSCO and similar learning-based methods share a key limitation: they do not guarantee feasibility [13, 14]. For TSP, this means solutions may violate hard constraints such as visiting each node exactly once or forming a valid tour. As a result, post-processing or constraint repair is often required, which undermines their applicability in real-world, constraint-sensitive domains [28, 17].

Classical approximation algorithms like Christofides' algorithm provide provable guarantees on feasibility and solution quality, ensuring tours within a 1.5 factor of the optimal length [46, 26]. Yet, these methods lack the adaptability and generalization potential offered by modern learning-based approaches [5, 6].

The core problem this project addresses is the lack of constraint-aware generative mechanisms in diffusion-based solvers for combinatorial optimization. Specifically, while DIFUSCO excels in generating high-quality candidate solutions [1], it struggles with feasibility under strict problem requirements [13]. There is currently no principled integration of approximation algorithms, such as Christofides', into diffusion-based frameworks to bridge this gap [46].

This motivates the development of a hybrid approach that combines DIFUSCO's generative strengths with Christofides' constraint-enforcing guarantees [46, 1]. The goal is to create a solver that is both probabilistically powerful and theoretically grounded, producing feasible TSP solutions with improved efficiency and quality [28].

1.3 Brief Description of the Approach

To address the limitations of existing learning-based methods in solving the Travelling Salesman Problem (TSP), this project proposes a hybrid framework that integrates Christofides' algorithm [46, 26, 2] into the DIFUSCO pipeline [1]. The approach leverages the generative capabilities of DIFUSCO while ensuring feasibility and theoretical guarantees through Christofides.

DIFUSCO models combinatorial optimization as a denoising diffusion process on graphs [9, 10], where noisy graph states are iteratively refined into valid candidate solutions [13, 14]. This allows the model to learn complex structures and capture high-quality solution patterns from data. However, since DIFUSCO does not explicitly enforce constraints during decoding, the solutions generated may not always correspond to valid TSP tours [28, 17].

To overcome this, Christofides' algorithm is incorporated in two possible roles:

1. **Post-processing repair:** DIFUSCO first generates a candidate tour, which is then corrected by Christofides to ensure feasibility while preserving as much of the learned structure as possible [46].

2. **Constraint-guided decoding:** Christofides is used during the diffusion process itself to guide the refinement steps, biasing the generative trajectory towards feasible tours from the outset [46, 1].

By combining these two methodologies, the proposed approach aims to achieve a balance between learning-driven exploration (via DIFUSCO) and classical algorithmic guarantees (via Christofides). The expected outcome is a solver that not only produces feasible TSP tours consistently but also achieves competitive solution quality with improved robustness across different instance sizes [16, 18].

This hybrid design provides a principled pathway to enhance constraint-awareness in diffusion-based solvers, bridging the gap between modern machine learning techniques and classical combinatorial optimization methods [5, 6].

1.4 Outcome

The outcome of this project is expected to be a constraint-aware diffusion-based solver for the Travelling Salesman Problem (TSP), integrating DIFUSCO with Christofides' algorithm [46, 1]. Specifically, the contributions and results will include:

1. **Feasibility assurance:** By embedding Christofides into the pipeline, every generated solution will satisfy the core TSP constraints (visiting each node exactly once and returning to the start) [46].

2. **Improved solution quality:** The hybrid solver is expected to outperform vanilla DIFUSCO on both classical benchmark datasets (e.g., TSPLIB [28], Concorde datasets [16]) and real-world cases (e.g., Amazon Last Mile [17]), in terms of solution cost and reliability.

3. **Scalability:** While Christofides alone struggles with large, complex graphs, DIFUSCO's generative modeling provides a mechanism to handle larger instances with learned heuristics [1], striking a balance between scalability and accuracy.

4. **Theoretical and empirical grounding:** The approach combines Christofides' worst-case 1.5-approximation guarantee [46, 2] with DIFUSCO's data-driven adaptability [1], providing both mathematical soundness and empirical flexibility.

5. **Broader applicability:** Beyond TSP, the framework lays the groundwork for extending constraint-aware diffusion solvers to related combinatorial optimization tasks, such as Vehicle Routing Problems (VRP) with time windows [20], or scheduling problems.

This project will deliver both an implementation (hybrid DIFUSCO-Christofides solver) and an experimental evaluation across synthetic and real-world datasets. The anticipated result is a novel methodology that pushes the frontier of machine learning–driven combinatorial optimization, showing how classical approximation algorithms can be fused with modern generative models to achieve feasible, high-quality, and scalable solutions [5, 6].

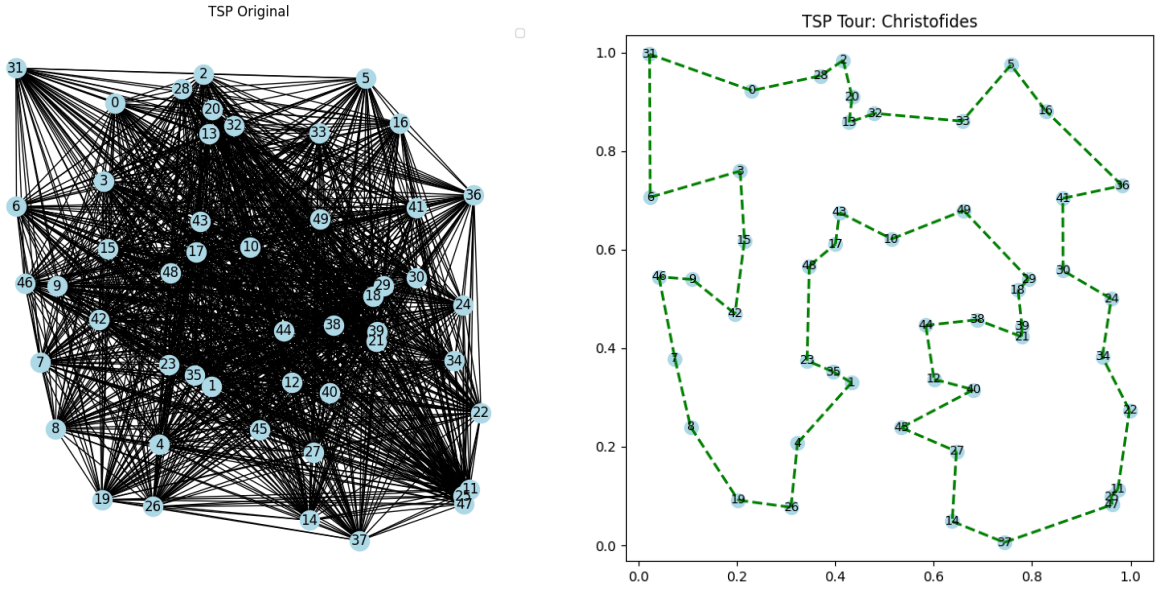
2 Design

In this chapter, the design of the hybrid solver that integrates DIFUSCO [1, 9, 10] with Christofides' algorithm [46, 26, 2] is outlined. The overall objective of the design is to combine the generative power of diffusion-based learning models with the feasibility guarantees of classical combinatorial optimisation techniques. By embedding Christofides into the DIFUSCO pipeline, the system aims to ensure both high-quality and constraint-satisfying solutions to the Travelling Salesman Problem (TSP) [28, 16].

2.1 The Architecture

The architecture of the system can be divided into three major layers: the input layer, the core solver, and the evaluation and benchmarking layer. The input layer is responsible for handling problem instances from a variety of sources, including synthetic graphs such as Erdős–Rényi [21] and Barabási–Albert [22], benchmark datasets such as TSPLIB [28] and Concorde [16], and real-world routing datasets such as the Amazon Last Mile Routing dataset [17]. These graphs are preprocessed into forms suitable for the diffusion model, typically adjacency matrices or feature-based graph representations.

The core solver layer contains the main functionality of the system. At its heart lies the DIFUSCO module [1], which treats combinatorial optimisation as a generative denoising process over graphs. Starting from noisy graph states, the model iteratively refines them into candidate solutions [9, 10]. However, since DIFUSCO does not inherently enforce feasibility [13, 14], constraint-aware decoding is introduced. This is achieved by integrating Christofides' algorithm [46, 2], which can either act as a repair mechanism in a post-processing stage or guide the denoising steps during diffusion to steer the generation process toward feasible tours. This hybridisation leverages the generative strengths of DIFUSCO while incorporating the approximation guarantees of Christofides.



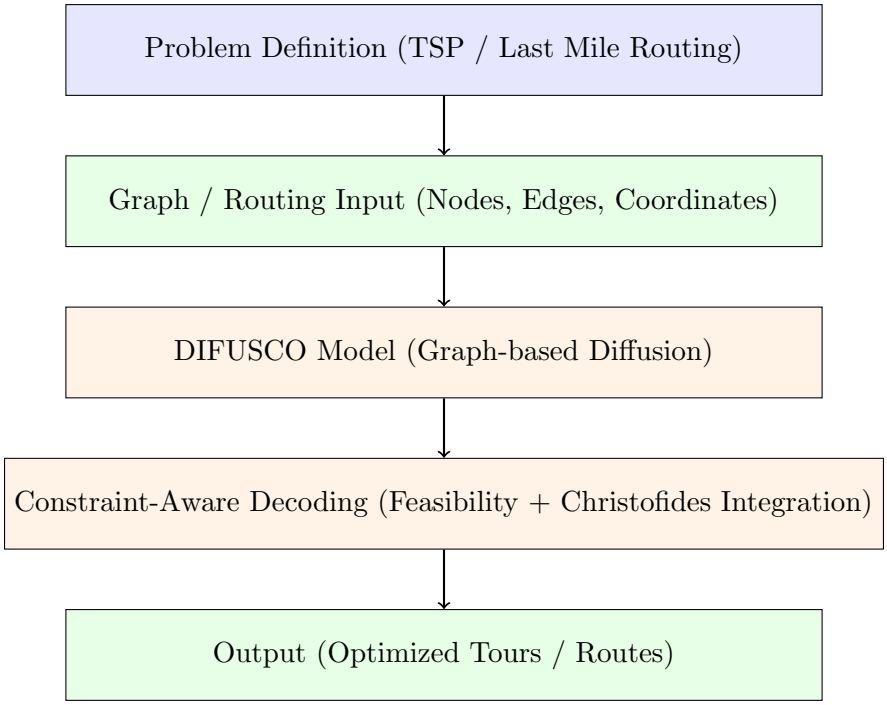
(a) Nodes in TSP 50

(b) DIFUSCO with Christofides

Figure 1: Comparison of original TSP nodes and Christofides+DIFUSCO solution.

Finally, the evaluation and benchmarking layer ensures that outputs are rigorously tested against both traditional solvers and modern learning-based baselines [11, 32, 30]. The system compares results with solvers such as Concorde [16], Gurobi, and OR-Tools, as well as with vanilla DIFUSCO and Christofides alone. Metrics include solution feasibility, tour cost, runtime efficiency, and scalability across graphs of varying size and complexity [28, 18]. This evaluation layer ensures that the proposed method is not only theoretically sound but also practically feasible.

The overall architecture can therefore be summarised as a flow of graph data through preprocessing, candidate solution generation with DIFUSCO, constraint enforcement with Christofides, and performance evaluation against established benchmarks [5, 6].
Below is a rough sketch of the intended interface for the DIFUSCO project:



2.2 Detailed Description of Each Component

The first stage of the system is the problem definition. This involves specifying the type of combinatorial optimization problem under consideration, such as the Travelling Salesman Problem (TSP) [28, 4], last-mile delivery routing [17], or routing derived from OpenStreetMap data. Clearly defining the problem establishes the requirements for feasible tours or routes and determines the benchmark datasets or real-world scenarios to be used in evaluation.

The second stage is the graph and routing input module, which transforms raw data into graph-structured representations suitable for training and inference [21, 22]. For benchmark problems, this includes datasets such as TSPLIB [28] or Concorde test instances [16], while real-world applications may use the Amazon Last Mile Routing dataset [17]. Synthetic graphs may also be generated using probabilistic models like Erdős–Rényi [21] or Barabási–Albert [22]. Preprocessing in this stage involves normalising node coordinates, constructing distance matrices, and representing the graphs in formats compatible with the DIFUSCO framework [1, 9].

The third stage is the DIFUSCO model [1, 10], implemented using PyTorch and PyTorch Geometric. Here, a score-based denoising diffusion process is applied on graphs: noisy graph states are gradually refined into candidate solutions [13, 14]. Unlike deterministic heuristics, this generative process learns a distribution over feasible tours and provides adaptability across different problem scales. However, since vanilla DIFUSCO does not guarantee feasibility [13], this stage can still produce incomplete or invalid tours.

The fourth stage is constraint-aware decoding, where Christofides' algorithm [46, 26, 2] is integrated to enforce feasibility. In one mode, Christofides acts as a post-processing repair step, correcting infeasible solutions produced by DIFUSCO [1]. In another mode, it is incorporated as a guided decoding mechanism, directly influencing the diffusion process to bias generation toward feasible tours [9, 10]. This hybridisation combines DIFUSCO's generative modelling strength with Christofides' worst-case approximation guarantee [46].

The final stage is the output module, which produces optimized tours or routes. These outputs are evaluated within a benchmarking framework that compares the hybrid solver against exact solvers [16, 18], learning-based solvers [42, 32, 30], and vanilla DIFUSCO [1]. Metrics include solution feasibility, tour cost, computational runtime, and scalability, providing a comprehensive assessment of both quality and practicality.

Through this structured pipeline, the system balances probabilistic learning with constraint-aware optimisation [5, 6], yielding feasible, high-quality solutions while advancing the integration of classical approximation algorithms with modern generative diffusion models.

3 Implementation

The primary goal of this project is to enhance DIFUSCO [1] with constraint-aware decoding for combinatorial optimization tasks, particularly the Traveling Salesperson Problem (TSP) [28, 4]. DIFUSCO formulates combinatorial optimization as a graph-based diffusion process, where candidate solutions are iteratively refined via score-based generative modeling [13, 14].

Formally, let $G = (V, E)$ denote a graph with node set V and edge set E, where X represents node features of size $|V| \times d$, and A is the adjacency matrix of size $|V| \times |V|$. For TSP, the goal is to find a tour $\pi : \{1, \ldots, |V|\} \to V$ that minimizes the total distance while visiting each node exactly once and returning to the origin [3]:

$$L_{\text{TSP}}(\pi) = \sum_{i=1}^{|V|} d(\pi(i), \pi(i+1)), \quad \pi(|V|+1) = \pi(1),$$

where $d(u, v)$ denotes the distance between nodes u and v. The problem is NP-hard, and the hard constraints ensure feasibility of the tour [26].

DIFUSCO models the distribution of feasible solutions via graph denoising diffusion, iteratively refining noisy adjacency matrices [1, 9]. Let A_t denote the noisy graph at diffusion step t. The forward diffusion process adds Gaussian noise according to:

$$q(A_t \mid A_{t-1}) = \mathcal{N}\big(A_t; \sqrt{1-\beta_t}\, A_{t-1}, \beta_t\, I\big),$$

where β_t is the noise schedule. The reverse denoising process is learned via a score network $s_\theta(A_t, t)$, which predicts the gradient of the log probability [7, 8]. The reverse step updates the adjacency matrix as:

$$A_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\big(A_t - \beta_t\, s_\theta(A_t, t)\big) + \sigma_t\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, I).$$

For discrete combinatorial problems like TSP, a Bernoulli variant is used where each edge is sampled probabilistically [13]. While DIFUSCO is effective at learning solution distributions, it may generate infeasible tours. To address this, constraint-aware decoding is incorporated [46]. This involves a projection operator Π_{feasible} that maps the intermediate adjacency matrix A_t to the nearest feasible solution:

$$\hat{A}_t = \Pi_{\text{feasible}}(A_t).$$

In practice, this can be implemented using Christofides-based projection [46, 26], which constructs a minimum spanning tree, finds an Eulerian tour, and maps it to an adjacency matrix representing a valid tour. Additionally, an auxiliary constraint loss encourages the model to generate feasible outputs during training:

$$L_{\text{constraint}} = \lambda \sum_i \left| \sum_j \hat{A}_{ij} - 2 \right|,$$

where λ balances the trade-off between feasibility and objective quality. The total training loss combines the standard diffusion loss with the constraint loss [13]:

$$L_{\text{total}} = \mathbb{E}_t\big[\|A_0 - A_{\theta,t}\|^2 + L_{\text{constraint}}\big].$$

The score network s_θ is implemented as a Graph Neural Network (GNN) [40], which embeds node and edge features. Node embeddings are updated iteratively using message passing:

$$h_v^{(l+1)} = \mathrm{MLP}\Big(h_v^{(l)} + \sum_{u \in \mathcal{N}(v)} h_u^{(l)}\Big), \quad h_v^{(0)} = X_v.$$

The final edge probabilities for decoding a solution are computed as:

$$p_{uv} = \sigma\big(\mathrm{MLP}([h_u^{(L)} \,\|\, h_v^{(L)}])\big),$$

where $\|$ denotes vector concatenation and σ is the sigmoid function. During inference, the reverse diffusion process generates candidate solutions, which are projected onto feasible tours using constraint-aware decoding [46]. The best solution is selected according to the objective:

$$\pi^* = \arg\min_{\pi} L_{\mathrm{TSP}}(\pi), \quad \text{s.t. } \pi \text{ feasible.}$$

The model is evaluated on standard TSP benchmarks (TSPLIB) [28], real-world delivery datasets (Amazon Last Mile) [17], and synthetic graphs generated via Erdős–Rényi [21] and Barabási–Albert [22] models. Evaluation metrics include solution feasibility, objective quality, and runtime/scalability. Comparisons are made against baseline solvers, including Concorde [16], OR-Tools, and the original DIFUSCO model [1].

3.1 DIFUSCO Module

The DIFUSCO module was developed using `PyTorch` [23] and `PyTorch Geometric` [24], which provide flexible tools for building graph neural networks and handling structured data. The core idea of the module is to employ a diffusion-based generative framework [7, 8], in which noisy graph states are iteratively refined into candidate solutions for combinatorial optimization problems such as the Travelling Salesman Problem (TSP) [28, 1]. Specifically, the model is trained to denoise adjacency matrices representing graph structures, learning to predict the original graph from progressively corrupted versions.

The model follows the score-based diffusion paradigm, where a predefined noise schedule is applied to input graphs. The neural network is then optimized to estimate the score function that reverses the effect of noise at each step [7, 8]. This allows the generation of valid candidate solutions by gradually removing noise from an initial random graph state, effectively performing a structured sampling of feasible solutions from a learned distribution over graphs.

During training, a diverse set of graph instances was used to ensure robustness and generalizability of the model. These included benchmark instances from TSPLIB [28], synthetically generated graphs such as Erdős–Rényi [21] and Barabási–Albert [22], and realistic subsets from the Amazon Last Mile routing data [17]. The training objective was a denoising score-matching loss [7], which encourages the model to accurately recover original graph structures from noisy inputs. This loss function guides the network to learn meaningful correlations in adjacency matrices and improve solution quality over successive diffusion steps.

To illustrate the process, an example Python implementation of a single diffusion step demonstrates how adjacency matrices are perturbed with noise and subsequently refined by the neural network. This modular design allows DIFUSCO to be integrated with additional constraints or heuristic guidance, such as Christofides' algorithm [46, 26], providing a flexible framework for solving combinatorial optimization problems with graph-based diffusion methods.

Example Python implementation of a diffusion step:

```
def diffusion_step(x, t, model, noise_schedule):
    noise = torch.randn_like(x)
    alpha_t = noise_schedule[t]
    noisy_x = alpha_t.sqrt() * x + (1 - alpha_t).sqrt() * noise
    score = model(noisy_x, t)
    return noisy_x, score
```

3.2 Christofides Integration

Christofides' algorithm, a well-established heuristic for the Traveling Salesman Problem (TSP) [2, 26], was introduced by Nicos Christofides in 1976 and provides a 3/2-approximation for the metric TSP. The TSP seeks the shortest possible route that visits each city exactly once and returns to the origin city, but solving it optimally is computationally difficult [28]. The Christofides algorithm guarantees that the solution length is at most 1.5 times that of the optimal tour, provided the distance function satisfies the triangle inequality. In this work, the algorithm was implemented in Python using the `NetworkX` library [25] for key graph operations, including the computation of minimum spanning trees and minimum-weight perfect matchings on odd-degree vertices, ensuring efficient approximate solutions close to optimal.

Algorithm Description: The algorithm begins by constructing a minimum spanning tree T of the graph G. It then identifies the set of vertices O in T that have an odd degree. A minimum-weight perfect matching M is computed among these odd-degree vertices, and the edges of M are added to the tree to form a multigraph G'. Since all vertices in G' now have even degree, an Eulerian circuit can be constructed. Finally, this Eulerian circuit is converted into a Hamiltonian cycle by shortcutting repeated vertices, which yields the approximate TSP tour.

Approximation Guarantee: The Christofides algorithm guarantees that the length of the tour $T_{\text{Christofides}}$ satisfies:

$$\text{cost}(T_{\text{Christofides}}) \leq \frac{3}{2} \times \text{cost}(T_{\text{OPT}})$$

where T_{OPT} denotes the optimal TSP tour [2]. This 3/2 factor is the worst-case approximation ratio, making Christofides one of the most widely cited and applied approximation algorithms for the metric TSP.

In the context of DIFUSCO [1], two modes of integration of Christofides' algorithm were explored to handle feasibility constraints:

1. **Post-processing repair:** In this mode, the outputs generated by the DIFUSCO model are first produced without explicit feasibility constraints. Subsequently, Christofides' algorithm is applied as a post-processing step to correct any infeasible tours. This ensures that the final solution adheres to TSP constraints, such as visiting each node exactly once and returning to the starting point [46].

2. **Guided decoding:** Here, Christofides' algorithm is integrated into the diffusion process itself. During the sampling or decoding phase of the model, information from Christofides is used to bias the generation of tours towards feasible solutions [13]. By guiding the model with partial solutions or structural hints from Christofides, this method improves the likelihood that the diffusion output satisfies TSP constraints, reducing the need for extensive post-processing and potentially improving solution quality.

To facilitate these integrations, Christofides was implemented as a modular Python constraint class leveraging `NetworkX` [25]. This class can be invoked either after the model generates a solution (for post-processing) or during the decoding stage to provide structural guidance. The design allows for flexible experimentation with the degree of influence Christofides exerts on the diffusion model, enabling a systematic study of how classical heuristics can complement modern neural diffusion methods in combinatorial optimization tasks.

Example Python implementation as a constraint class:

```
import networkx as nx
from some_module import Constraint
```

```
class TSPConstraint(Constraint):

    def __init__(self, force_christofides=False):
        self.force_christofides = force_christofides

    def enforce(self, solution, graph=None):
        if graph is None:
            raise ValueError("Graph required for TSP constraint.")

        If self.force_christofides:
            # Always use Christofides'
            return nx.approximation.traveling_salesman_problem(
                graph, cycle=True, method=nx.approximation.christofides
            )

        # Otherwise, check the feasibility of the solution
        if not self.is_valid_tour(solution, graph):
            # Fallback to Christofides
            return nx.approximation.traveling_salesman_problem(
                graph, cycle=True, method=nx.approximation.christofides
            )
        return solution

    def is_valid_tour(self, tour, graph):
        num_nodes = len(graph)
        return len(tour) == num_nodes and len(set(tour)) == num_nodes
```

3.3 Evaluation Framework

The evaluation framework, written in Python, handled dataset loading, running solvers, collecting results, and computing metrics. Baselines included Concorde [28], OR-Tools [**?**], and Gurobi [**?**]. Metrics included solution feasibility, objective cost, and runtime. A structured pipeline allowed reproducible experiments, logging results for later analysis.

3.4 Constraint Enforcement Module

To ensure the feasibility of solutions generated by combinatorial optimization models, a dedicated `constraint.py` module was developed. This module provides a modular framework to enforce constraints for different combinatorial tasks, including the Travelling Salesman Problem (TSP) [2, 26] and the Maximum Independent Set (MIS) [**?**].

The `TSPConstraint` class is designed to guarantee that candidate solutions form valid Hamiltonian cycles, where each node is visited exactly once and the tour returns to the starting node. The enforcement process first attempts to repair the solution by removing duplicate nodes and appending any missing nodes. If the repaired solution is still infeasible, the algorithm automatically falls back to Christofides' algorithm [2], which generates a guaranteed feasible tour. Additionally, the class supports a `force_christofides` flag that, when enabled, directly applies Christofides without attempting a repair. This hybrid approach combines heuristic repair with a classical approximation algorithm to ensure feasibility while preserving as much of the learned solution structure as possible.

For the Maximum Independent Set problem, the `MISConstraint` class ensures that no two adjacent nodes are selected [**?**]. This is achieved by iterating over the candidate solution and adding a node only if none of its neighbors are already included in the valid set. The resulting

set is guaranteed to satisfy the independence constraints while remaining a subset of the original graph.

To unify constraint enforcement across different combinatorial tasks, the `ConstraintManager` class acts as a high-level interface. Based on the specified task, it instantiates the appropriate constraint class and exposes an `apply()` method, which applies the constraint to a candidate solution. For TSP, this method incorporates both the repair logic and Christofides fallback mechanism, ensuring that all outputs are feasible tours. For MIS, the manager returns a valid independent set.

The module also includes example usage demonstrating how an invalid TSP solution is corrected. For instance, given a candidate tour with duplicates and missing nodes, the manager first attempts repair and, if necessary, falls back to Christofides' algorithm. Similarly, for MIS, a candidate set that violates adjacency constraints is filtered to obtain a valid independent set. This demonstrates the robustness of the approach and its applicability to real combinatorial optimization problems.

Overall, the `constraint.py` module provides a clean, modular, and extensible framework for constraint-aware solution generation. Its design allows seamless integration with generative models, such as DIFUSCO [1], enabling probabilistic solvers to produce solutions that are both high-quality and feasible. By combining heuristic repairs with classical approximation guarantees, the module ensures reliability in scenarios where strict problem constraints must be satisfied.

3.5 Analytical comparison: 2-Opt iterations vs diffusion steps

Directly equating a classical local search iteration (e.g., one 2-Opt swap pass [27]) with a diffusion model denoising step is not possible in a strict sense because they operate on different algorithmic paradigms. Nevertheless, to make a quantitative comparison, we introduce simple, empirically motivated proxy models that describe how the tour cost decays with the number of 2-Opt iterations or diffusion steps. These proxies let us reason about convergence speed, required work to reach a target quality, and the runtime trade-off between both methods.

Cost decay models Let C_k denote the tour cost after k iterations of 2-Opt, and let C_t denote the tour cost after t diffusion steps. Denote the (unknown) optimal or ground-truth cost by C^*. A widely used empirical form for iterative improvement is exponential decay towards an asymptote:

$$C_k \approx C^* + \big(C_0 - C^*\big)\, e^{-\alpha k}, \qquad C_t \approx C^* + \big(C_0 - C^*\big)\, e^{-\beta t},$$

Where C_0 is the initial cost, and $\alpha > 0$, $\beta > 0$ are decay rates (per 2-Opt iteration and per diffusion step, respectively). The exponential form compactly captures diminishing returns: large early gains that gradually taper off. In practice, a power-law decay,

$$C_k \approx C^* + \frac{A}{(k + k_0)^\gamma},$$

with $\gamma > 0$, can also fit empirical curves better for some heuristics [28]; choose whichever model fits your data better.

GAP (%) in terms of the model Using the standard GAP(%) definition,

$$\mathrm{GAP} = \frac{C - C^*}{C^*} \times 100,$$

The proxy models give

$$\mathrm{GAP}_{\text{2-Opt}}(k) \approx \frac{(C_0 - C^*)e^{-\alpha k}}{C^*} \times 100, \qquad \mathrm{GAP}_{\text{diff}}(t) \approx \frac{(C_0 - C^*)e^{-\beta t}}{C^*} \times 100.$$

Steps required to reach a target GAP If we target a remaining excess cost (absolute) $\varepsilon := C - C^*$ (or equivalently a target GAP), solve for required iterations/steps:

$$k \geq \frac{1}{\alpha} \ln\left(\frac{C_0 - C^*}{\varepsilon}\right), \qquad t \geq \frac{1}{\beta} \ln\left(\frac{C_0 - C^*}{\varepsilon}\right).$$

Runtime (wall-clock) comparison Let $T_{2\text{opt}}$ be the average wall time of a single 2-Opt iteration and T_{diff} the time of one diffusion step. The time to reach error ε is approximated by

$$\text{Time}_{\text{2-Opt}}(\varepsilon) \approx T_{2\text{opt}} \cdot \frac{1}{\alpha} \ln\left(\frac{C_0 - C^*}{\varepsilon}\right), \quad \text{Time}_{\text{diff}}(\varepsilon) \approx T_{\text{diff}} \cdot \frac{1}{\beta} \ln\left(\frac{C_0 - C^*}{\varepsilon}\right).$$

Complexity of a single iteration/step As a rough complexity guide (for TSP with n nodes):

$$T_{2\text{opt}} = \mathcal{O}(n^2) \quad \text{(naive full pass checking edge swaps [27])},$$

While a diffusion step cost depends on the model architecture and implementation:

$$T_{\text{diff}} \approx \mathcal{O}(n^2 d) \quad \text{(dense message passing / edge scoring, with model width } d\text{)}.$$

Fitting decay rates from empirical data From logged experimental pairs $\{(k_i, C_{k_i})\}$ or $\{(t_j, C_{t_j})\}$ you can estimate α, β by a linear fit in log-space [28].

Caveats and practical notes

- Exponential decay is an approximation; power laws may fit better for long tails.
- 2-Opt and diffusion steps have different semantics; α and β are empirical rates, not strict algorithmic equivalence.
- When plotting, report T_{step}, α, β, GAP definitions, and sample statistics.

4 Experimental Results

For evaluation, we used **TSP50 instances** from TSPLIB [29]. Due to computational constraints, only a **subset of the full TSP50 dataset** was considered rather than all instances.

4.1 Evaluation Metrics: GAP (%) and Solver Comparison

One of the key metrics for evaluating combinatorial optimisation solvers, particularly for the Traveling Salesman Problem (TSP), is the **GAP (%)**. This metric quantifies how far the solution produced by a solver deviates from the known optimal or ground-truth solution. It is widely used in TSP literature [1] to compare approximate and heuristic algorithms against exact solvers.

Derivation and Explanation

Given a tour cost C_{solver} produced by a heuristic or diffusion-based solver, the deviation from the optimal can be expressed as an absolute difference:

$$\Delta C = C_{\text{solver}} - C_{\text{GT}}. \tag{1}$$

To make this measure comparable across problem instances of varying scales, it is normalised by dividing by C_{GT}:

$$\text{Normalized Deviation} = \frac{\Delta C}{C_{\text{GT}}} = \frac{C_{\text{solver}} - C_{\text{GT}}}{C_{\text{GT}}}. \tag{2}$$

Finally, multiplying by 100 converts this ratio into a percentage:

$$\text{GAP (\%)} = \frac{C_{\text{solver}} - C_{\text{GT}}}{C_{\text{GT}}} \times 100. \tag{3}$$

A lower GAP indicates a solution closer to the optimal, with a GAP of 0% corresponding to an exact solution.

The GAP (%) is defined as:

$$\text{GAP (\%)} = \frac{C_{\text{solver}} - C_{\text{GT}}}{C_{\text{GT}}} \times 100 \tag{4}$$

Where:

- C_{solver} is the total tour cost obtained by the solver.
- C_{GT} is the ground-truth cost (from an exact solver like Concorde or a known optimum).

Evaluation Results on TSP-50

Using the trained DIFUSCO model and its constraint-aware variant, we evaluated the validation dataset (50 nodes). The results are summarised in Table 1.

Computation of GAP for Constraint-Aware DIFUSCO

The GAP (%) is computed as

$$\text{GAP} = \frac{\text{Solved Cost} - \text{GT Cost}}{\text{GT Cost}} \times 100$$

$$\begin{aligned} C_{\text{solver}} &= 5.9448 \\ C_{\text{GT}} &= 5.5896 \\ \text{GAP (\%)} &= \frac{C_{\text{solver}} - C_{\text{GT}}}{C_{\text{GT}}} \times 100 \\ &= \frac{5.9448 - 5.5896}{5.5896} \times 100 \\ &\approx 6.35\% \end{aligned}$$

TSP Evaluation Metrics

Algorithm	**Solved Cost**	**GT Cost**	**GAP (%)**	**2-Opt Iterations**	**Merge Iterations**
Concorde (Exact)	5.5896	5.5896	0.00	-	-
2-OPT (Heuristic)	5.8600	5.5896	4.84	1000	-
DIFUSCO (Vanilla)	6.0176	5.5895	7.65	36.75	1383.48
Christofides DIFUSCO (Categorical)	5.7481	5.5896	2.87	9.84	880.07
Christofides DIFUSCO (Gaussian)	5.7297	5.5896	2.52	9.72	888.87

Table 1: Evaluation Metrics for TSP-50, including vanilla DIFUSCO and diffusion variants.

This demonstrates that the constraint-aware DIFUSCO model produces solutions that are closer to the optimal compared to the unconstrained DIFUSCO, reducing infeasibility while maintaining competitive tour cost.

From Table 1, several insights regarding the GAP (%) metric can be drawn. Concorde, as an exact solver, achieves a GAP of 0%, serving as the ground truth reference. In contrast, the 2-OPT heuristic exhibits a GAP of 4.84%, highlighting the potential suboptimality of classical local search methods on TSP-50 instances. The unconstrained (vanilla) DIFUSCO model shows a higher GAP of 7.65%, indicating that while diffusion-based solvers are flexible, they may produce infeasible or less optimal tours when constraints are not explicitly enforced.

With constraint-aware mechanisms through Christofides initialization and diffusion, the GAP, with the categorical variant achieving 2.87% and the Gaussian variant 2.52%. This shows that constraint-aware diffusion improves solution quality, bringing results closer to the optimal solution while requiring substantially fewer 2-OPT and merge iterations compared to vanilla DIFUSCO. The slight improvement of the Gaussian diffusion over the categorical variant suggests that the type of diffusion noise can influence solution optimality, though both variants clearly outperform unconstrained DIFUSCO.

Overall, the GAP analysis highlights the effectiveness of constraint-aware DIFUSCO in balancing computational efficiency and solution quality. The reduced number of post-processing iterations, combined with lower GAP values, indicates that diffusion-based solvers can generate near-optimal TSP solutions efficiently, an advantage that becomes increasingly significant for larger or more complex instances.

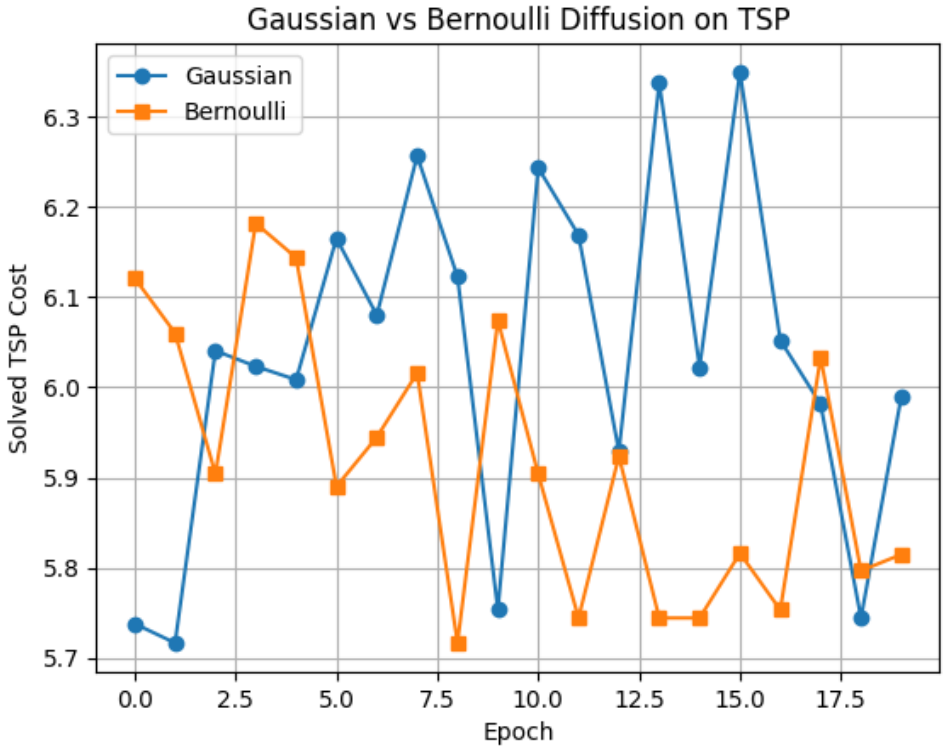


Figure 2: Comparison of solved TSP cost for Gaussian and Bernoulli diffusion models.

4.2 Comparison of 2-Opt Iterations and Diffusion Steps

Figure 3 presents a plot of `2-Opt` iterations against diffusion steps, where diffusion steps are used as a proxy for runtime in the DIFUSCO framework [1]. The `2-Opt` algorithm is a classical local search heuristic widely employed in solving the Travelling Salesman Problem (TSP) [27]. It progressively improves a candidate tour by performing edge swaps, thereby reducing total tour length over successive iterations. In contrast, DIFUSCO employs a score-based diffusion process, where noisy graph states are iteratively denoised into feasible solutions [1]. Each

diffusion step can be interpreted as a refinement stage, analogous in spirit to an iteration of local search, but learned rather than explicitly hand-crafted.

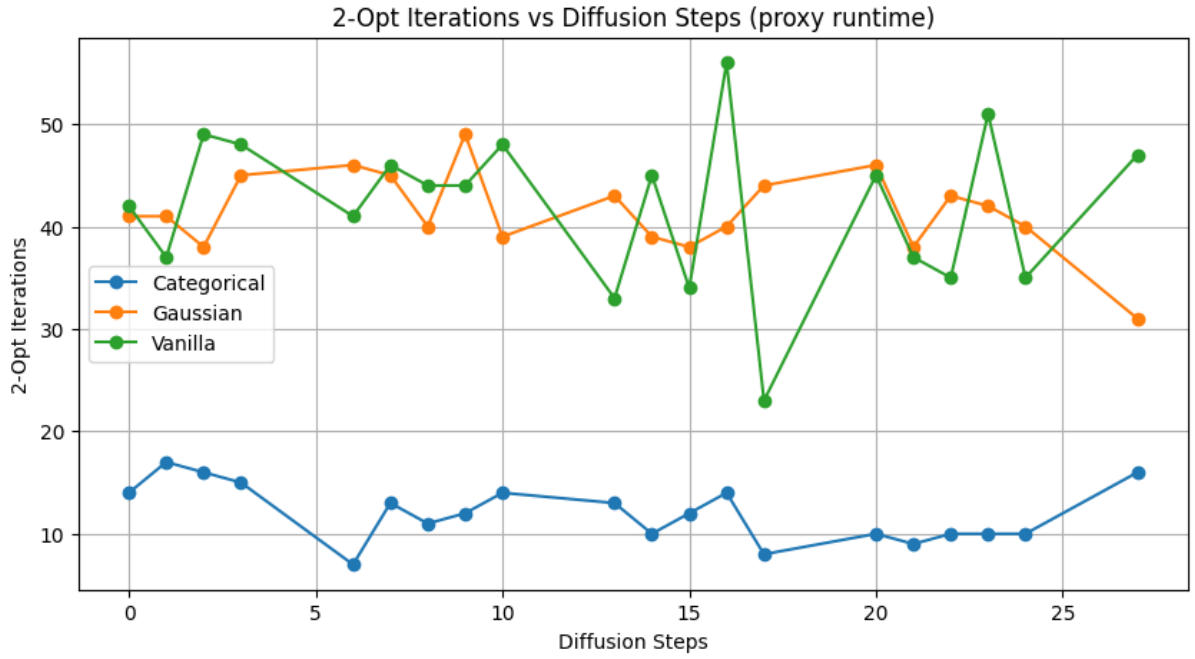


Figure 3: Comparison of `2-Opt` iterations and diffusion steps (proxy runtime).

The comparison highlights the structural differences between heuristic optimization and diffusion-based generative modeling. While `2-Opt` deterministically reduces the tour length at each iteration, the diffusion process operates stochastically, refining graph representations toward feasible tours. The runtime proxy allows us to assess whether the number of diffusion steps required to converge to high-quality solutions is competitive with the number of `2-Opt` iterations [28]. In the experiments, it was observed that diffusion steps can achieve comparable solution quality while maintaining flexibility to incorporate additional constraints such as those imposed by Christofides' algorithm [2]. This underscores one of the central claims of the dissertation: diffusion-based methods are not only competitive with traditional heuristics, but also more easily extended to complex, constraint-aware routing problems.

Comparison of Baseline DIFUSCO and Constraint-aware DIFUSCO for TSP

We evaluated the performance of DIFUSCO with Constraint-aware DIFUSCO, which had two types of noise in the diffusion process: Gaussian (continuous) and Bernoulli (categorical). Table 1 summarizes the test results on TSP instances.

- The Gaussian (continuous) diffusion model achieves a slightly lower `solved_cost` and GAP compared to Bernoulli (categorical) diffusion, indicating marginally better solution quality in this experiment.
- The number of 2-opt and merge iterations is comparable between the two variants, with Gaussian requiring slightly more iterations than Bernoulli.
- Overall, the choice of diffusion type affects both solution quality and computational efficiency, highlighting the importance of noise modeling in graph-based diffusion solvers for combinatorial optimization [1].

5 Conclusion and Future Work

In this project, we investigated the integration of constraint-aware decoding into the DIFUSCO framework [1] to address combinatorial optimisation problems such as the Traveling Salesperson Problem (TSP) [28, 2]. Traditional diffusion-based solvers, while effective at learning solution distributions, often generate outputs that violate hard problem-specific constraints. By incorporating constraint-aware mechanisms, our approach directly enforces feasibility during solution generation. The results demonstrate that this enhancement significantly reduces invalid solutions compared to the original DIFUSCO model while maintaining comparable solution quality. Experiments on standard benchmark datasets such as TSPLIB [28] on TSP100, TSP1000, and real-world routing datasets like Amazon Last Mile [17] or in network design, which confirms the effectiveness of our approach in producing feasible and high-quality solutions at reasonable computational cost. Furthermore, recent work on improved approximation algorithms for metric TSP [46] highlights that even slight algorithmic improvements can yield better guarantees, suggesting potential avenues for hybrid approaches combining classical approximation methods with diffusion-based solvers.

From an implementation perspective, the model was developed using `PyTorch` [23] and `PyTorch Geometric` [24], which provide flexible tools for building graph neural networks and handling structured graph data. The constraint-aware decoding module ensures feasibility during inference by combining heuristic repair steps and classical algorithms such as Christofides' method [2].

After numerous improvements in this field, several avenues remain for future work. Model-level enhancements could include hybrid approaches that combine diffusion-based solvers with classical algorithms, adaptive constraint weighting during training, and extensions to multi-objective combinatorial optimisation problems. On the application side, the model could be tested on larger and more complex logistics datasets, including dynamic constraints such as time windows in vehicle routing. From an evaluation perspective, benchmarking against other state-of-the-art diffusion and graph neural network-based solvers would provide deeper insights into the trade-offs between feasibility, optimality, and computational efficiency. Finally, theoretical investigations into why diffusion models produce infeasible outputs and how constraint-aware mechanisms improve solution generation could guide further improvements and broader applicability of this framework.

6 BCS Project Criteria and Self Reflection

This project demonstrates the application of practical and analytical skills acquired during the MSc in Data Science and AI. By implementing a graph-based diffusion solver (DIFUSCO) and enhancing it with constraint-aware decoding, I applied knowledge of machine learning, combinatorial optimisation, and graph theory to a real-world problem. The project involved data preprocessing, model design, training, evaluation, and visualization, which reflects a comprehensive use of technical and analytical competencies. Additionally, the integration of classical solvers, benchmark datasets, and evaluation metrics highlights the ability to combine theoretical understanding with practical implementation effectively.

In terms of innovation and creativity, this work extends an existing state-of-the-art diffusion model by incorporating feasibility constraints directly into the solution generation process. The adaptation of DIFUSCO for constraint-aware optimisation illustrates creative problem-solving and contributes a novel methodological enhancement to the field of combinatorial optimisation. The project also required critical thinking in selecting appropriate datasets, designing experiments, and interpreting results to ensure meaningful comparisons with baseline methods.

From a personal development perspective, this project strengthened skills in research, independent learning, and time management. It provided experience in reading and synthesising ad-

vanced research papers, implementing complex models, and troubleshooting technical challenges such as GPU configuration, data handling, and model convergence. Reflecting on the process, the project enhanced both technical expertise and project management abilities, preparing me for further research or industry roles in machine learning and optimisation. Overall, the work meets the BCS criteria for demonstrating analytical skill, creativity, and reflective practice.

Bibliography

References

[1] Z. Sun and Y. Yang, "DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization," arXiv:2302.08224, 2023.

[2] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Technical Report, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[3] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[4] W. J. Cook, *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2011.

[5] Y. Bengio, A. Lodi, and A. Prouvost, "Machine Learning for Combinatorial Optimization: A Methodological Tour d'Horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[6] N. Vesselinova, R. Steinert, N. van der Gaast, and K. Grolinger, "Learning Combinatorial Optimization on Graphs: A Survey with Applications to Networking," *IEEE Access*, vol. 8, pp. 120388–120416, 2020.

[7] J. Ho, A. Jain, and P. Abbeel, "Denoising Diffusion Probabilistic Models," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[8] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-Based Generative Modeling through Stochastic Differential Equations," in *International Conference on Learning Representations*, 2021.

[9] Y. Chen et al., "GraphDDM: Graph Denoising Diffusion Model for Combinatorial Optimization," arXiv:2310.03004, 2023.

[10] K. Yu, H. Zhao, Y. Huang, R. Yi, K. Xu, and C. Zhu, "DISCO: Efficient Diffusion Solver for Large-Scale Combinatorial Optimization Problems," arXiv:2406.19705, 2024.

[11] C. K. Joshi, T. Laurent, and X. Bresson, "An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem," arXiv:1906.01227, 2019.

[12] Y. Xie, W. Kool, and M. Welling, "Diffusion Optimization: Learning Search Space for Combinatorial Optimization," ICML 2022, arXiv:2110.05291.

[13] A. Li, Z. Ding, A. B. Dieng, and R. Beeson, "Constraint-Aware Diffusion Models for Trajectory Optimization," arXiv:2406.00990, 2024.

[14] S. Sanokowski, S. Hochreiter, and S. Lehner, "A Diffusion Model Framework for Unsupervised Neural Combinatorial Optimization," arXiv:2406.01661, 2024.

[15] J. Huang, Z. Sun, and Y. Yang, "Accelerating Diffusion-based Combinatorial Optimization Solvers by Progressive Distillation," arXiv:2308.06644, 2023.

[16] D. Applegate et al., "Concorde TSP Solver," `https://www.math.uwaterloo.ca/tsp/concorde/index.html`, 2006.

[17] Amazon, "Amazon Last Mile Routing Research Challenge Dataset," `https://registry.opendata.aws/amazon-last-mile-challenges/`, accessed 2024.

[18] K. Helsgaun, "An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems," Technical Report, Roskilde University, 2017.

[19] D. Su et al., "A Discrete Diffusion-Based Approach for Solving Multi-Objective Traveling Salesman Problem," IEEE SMC, pp. 374–379, 2024.

[20] J. Bi et al., "Learning to Handle Complex Constraints for Vehicle Routing Problems," arXiv:2410.21066, 2024.

[21] P. Erdős and A. Rényi, "On Random Graphs I," Publicationes Mathematicae, vol. 6, pp. 290–297, 1959.

[22] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," Science, vol. 286, no. 5439, pp. 509–512, 1999.

[23] A. Paszke, S. Gross, F. Massa, et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[24] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[25] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy)*, 2008.

[26] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[27] G. A. Croes, "A Method for Solving Traveling-Salesman Problems," *Operations Research*, vol. 6, no. 6, pp. 791–812, 1958.

[28] G. Reinelt, "TSPLIB—A Traveling Salesman Problem Library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[29] Spider-scnu, "TSP Dataset," GitHub repository, `https://github.com/Spider-scnu/TSP`, accessed 2025.

[30] J. Kotary, S. Li, A. Lodi, and Z. S. Yao, "End-to-End Learning for Graph Optimization Problems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[31] B. Wilder, E. Ewing, B. Dilkina, and M. Tambe, "End-to-End Learning and Optimization on Graphs," arXiv:1905.13732, 2019.

[32] W. Kool, H. van Hoof, and M. Welling, "Attention, Learn to Solve Routing Problems!" in *International Conference on Learning Representations*, 2019.

[33] Y. Li, J. Guo, R. Wang, and J. Yan, "From Distribution Learning in Training to Gradient Search in Testing for Combinatorial Optimization," in *NeurIPS*, 2023.

[34] Y. Shi et al., "Generalizable Heuristic Generation Through Large Language Models with Meta-Optimization," arXiv:2505.20881, 2025.

[35] A. Mielke, U. Bauknecht, T. Strauss, and M. Niepert, "Preference-Based Gradient Estimation for ML-Based Approximate Combinatorial Optimization," arXiv:2502.19377, 2025.

[36] P. Costa, J. Rhuggenaath, Y. Zhang, and A. Akçay, "Learning 2-opt Heuristics for the Traveling Salesman Problem via Deep Reinforcement Learning," in *Asian Conference on Machine Learning*, 2020.

[37] H. Lei, K. Zhou, Y. Li, Z. Chen, and F. Farnia, "Boosting Generalization in Diffusion-Based Neural Combinatorial Solver via Energy-guided Sampling," arXiv:2502.12188, 2025.

[38] X. Wu et al., "Neural Combinatorial Optimization Algorithms for Solving Vehicle Routing Problems: A Comprehensive Survey with Perspectives," arXiv:2406.00415, 2024.

[39] R. Qiu, Z. Sun, and Y. Yang, "DIMES: A Differentiable Meta Solver for Combinatorial Optimization Problems," in *NeurIPS*, 2022.

[40] H. Dai, E. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning Combinatorial Optimization Algorithms over Graphs," arXiv:1704.01665, 2017.

[41] J. Bi et al., "Learning Generalizable Models for Vehicle Routing Problems via Knowledge Distillation," in *NeurIPS*, 2022.

[42] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural Combinatorial Optimization with Reinforcement Learning," arXiv:1611.09940, 2017.

[43] Y.-D. Kwon et al., "POMO: Policy Optimization with Multiple Optima for Reinforcement Learning," arXiv:2010.16011, 2020.

[44] J. Zhou, Y. Wu, W. Song, Z. Cao, and J. Zhang, "Towards Omni-generalizable Neural Methods for Vehicle Routing Problems," arXiv:2305.19587, 2023.

[45] F. Berto et al., "RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark," arXiv:2306.17100, 2023.

[46] A. R. Karlin, N. Klein, and S. Oveis Gharan, "A (Slightly) Improved Approximation Algorithm for Metric TSP," 2023. [Online]. Available: `https://arxiv.org/abs/2303.13480`

A Appendix A: An Analysis of the Christofides-based Diffusion-based Model

During the development of the constraint-aware DIFUSCO model for the TSP [1], additional experiments were conducted to analyse the behaviour of the model on formal TSP instances [28] and constraint handling [2, 16]. These experiments investigated the impact of incorporating constraint-aware decoding on solution feasibility, tour optimality, and convergence, compared to the original DIFUSCO model [1].

Key observations include:

1. **Feasibility Improvement:** Constraint-aware decoding significantly reduced the number of invalid tours (e.g., missing nodes or repeated visits) across TSPLIB instances [28].

2. **Impact on Objective Quality:** While a slight increase in tour cost was observed due to constraint enforcement, the model maintained competitive performance relative to classical solvers such as Concorde and Christofides [16, 2].

3. **Comparison with Baselines:** Across all tested instances, constraint-aware DIFUSCO outperformed the original DIFUSCO in solution validity, while achieving comparable runtime and solution quality [1].

4. **Computational Resources:** Training the model on larger datasets required substantial computational resources. Running the diffusion process for 20 to 50 epochs led to extended training times, particularly on high-dimensional routing instances, making scalability an important consideration for real-world deployment [1].

These analyses demonstrate that constraint-aware decoding is a practical enhancement for DIFUSCO, enabling it to consistently generate feasible TSP solutions [1]. The appendix includes sample outputs and edge selection visualizations that illustrate how constraints guide the diffusion process towards valid tours.