Solving Combinatorial Optimization Problems using Statistical Learning

by

Andrew Naguib

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Solving Combinatorial Optimization Problems using Statistical Learning

by

Andrew Naguib

Supervisory Committee

Dr. Waleed A. Yousef
Department of Electrical and Computer Engineering

Dr. Issa Traoré
Department of Electrical and Computer Engineering

Abstract

This thesis examines the use of geometric deep neural networks to provide competent solutions (in terms of runtime versus duality gap), not necessarily incumbent, to the capacitated vehicle routing problem and the bin packing problem—which have non-deterministic polynomial computational complexity.

The core idea is based on learning to approximate the decisions made by the branch and bound algorithm using the computationally expensive strong branching strategy. The classifiers - graph convolutional neural network, GraphSAGE, and graph attention network - are trained on six topologically different instances (to investigate the geographical dispersion effect on optimality) and evaluated on eight additional instances.

The experiments we conducted show that the proposed approach is able to match the performance of the branch and bound algorithm and improve upon it on two different branching strategies, while requiring significantly less computation time and explored branching nodes.

Table of Contents

Supervisory Committee ii

Abstract iii

Table of Contents iv

List of Figures v

Dedication vi

Acknowledgments vii

1 Prologue 1
- 1.1 Outline . . . 2
- 1.2 Summary of Research Contributions . . . 2
  - 1.2.1 Open-source Contributions . . . 2

I Foundations 4

2 Preliminaries and Literature Review 5
- 2.1 Background . . . 5
  - 2.1.1 Integer Program (IP) . . . 5
  - 2.1.2 Primal-Dual Method . . . 6
  - 2.1.3 Branch and Bound . . . 6
  - 2.1.4 Strong Branching Policy (SB) and Reliable Pseudo-Cost (RPC) . . . 7
  - 2.1.5 Capacitated Vehicle Routing Problem (CVRP) . . . 7
  - 2.1.6 Bin Packing Problem (BPP) . . . 9
- 2.2 Literature Review . . . 9

II Analysis 11

3 Geometrical Models for Combinatorial Optimization 12

3.1 The Routing Problem . . . 12
3.2 Conceptualization as a Supervised Learning . . . 14
3.3 Graph Convolutional Neural Network (GCNN) . . . 14
3.4 GraphSAGE . . . 16
3.5 Graph Attention Network (GAT) . . . 17
3.6 Experiments . . . 18
3.6.1 Datasets . . . 18
3.6.2 Training . . . 19
3.6.3 Evaluation . . . 20
3.7 Results . . . 21
3.7.1 Findings . . . 28
3.8 Discussion . . . 29
3.9 Technical Details . . . 30

4 Conclusion **31**

Bibliography **32**

List of Figures

2.1 An IP example consisting of two variables and three constraints. 5
2.2 The arcs on the dashed branches are equivalent to the δ_i, The bars hovering over each node is a *visual* representation of the upper bound. 7
2.3 A CVRP example with $n = 40$ and $k = 5$. Given the dispersed points shown in (a), CVRP asks to find the routes presented in (b). 8

3.1 portraying equations (2.1a)-(2.1b) as a bipartite graph; the corresponding integer program would then be $\sum \circ$; s.t. $\diamond$ 14
3.2 GraphSAGE accumulates information from a selection of neighboring nodes, with more distant nodes becoming increasingly influential in the aggregated information as the process continues. 16
3.3 GAT assigns attention scores to the relationships between each node and its neighboring ones. This is an example of a three-head attention mechanism. 17
3.4 Example on solving CVRP instance M-n151-k12 using both GraphSAGE and SCIP with allowed runtime of 8 hours. 19
3.5 The performance of the three CVRP classifiers against SCIP_{SB} calculated by (3.13), across the training instances. 23
3.6 The performance of the three CVRP classifiers against SCIP_{RPC} calculated by (3.13), across the training instances. 24
3.7 The average performance of each classifier across the CVRP evaluation instances; (a) compares against SCIP_{SB} and (b) compares against SCIP_{RPC}. . . 25
3.8 The performance of each classifier averaged by both of the CVRP training and evaluation instances against (a) SCIP_{SB} and (b) SCIP_{RPC}. 25
3.9 The performance of the three BPP classifiers against SCIP_{SB} calculated by (3.13), across the training instances. 26
3.10 The performance of the three BPP classifiers against SCIP_{RPC} calculated by (3.13), across the training instances. 26
3.11 The average performance of each classifier across the BPP evaluation instances; (a) compares against SCIP_{SB} and (b) compares against SCIP_{RPC}. . . 27
3.12 The performance of each classifier averaged by both of the BPP training and evaluation instances against (a) SCIP_{SB} and (b) SCIP_{RPC}. 27

To Mom and Dad.

Acknowledgments

I am grateful to everyone who enlightened me of the world and its complexities. Thanks to the many who provided me with love, tranquility, joy, and gaiety, for I will ensure it is paid forward.

I reflected on my mindset before meeting Prof. Waleed A. Yousef and upon which I realized how fragile it was. Prof. Yousef's courses instilled a unique vantage of science in me, and, even more strongly, enabled me to confidently delve into complex subjects. The close collaboration with him on several successful project has massively contributed to my development as a researcher and as a person. I am exceptionally grateful to the substantial investment of his time and energy to direct this work—thank you very much for moulding me into who I am today.

I am also indebted to Dr. Mohammad Alaggan for the conversations we had, which expanded my understanding of science and the vastness of it.

Thanks to ISOT laboratory, under the coordination of Prof. Traoré, for consistently supporting all activities that helped me improve as a researcher and providing an environment conducive to the intellectual freedom.

I am grateful to William Briguglio who made graduate school much more enjoyable than what I imagined—especially for our conversations on philosophy, theatre, and politics.

I am thankful to Ahmed Farrag who helped me getting my application complete for admission while I was still enrolled in the military service.

I also would like to thank the administrative team at UVic which smoothly processed all requests and documents, including Ashleigh Carlsen, Janice Closson, Kate Adams and Kelly Baker. In addition, I greatly appreciate the advice I received from Prof. Aaron Gulliver.

Chapter 1

Prologue

> Although all NP-complete problems share the same worst-case complexity, they have little else in common. When seen from almost any other perspective, they resume their healthy, confusing diversity. Approximability is a case in point
>
> *Chritos H. Papadimitriou*

This thesis revisits the pursue of solutions to combinatorial optimization problems, which are mostly NP-hard, or at least, the interesting ones of them. Approximation is the *de facto* reasonable approach given the infeasible attainment of optimality in practical settings—until a resolution is put to the $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ question.

Efficiently approximating solutions to NP-hard optimization problems has long been scrutinized; however, apart from isolated successes, the research path has stalled after realizing that the *hardness* could be the same as finding the optimal solution; in fact, unless P=NP, the approximation threshold, i.e., the greatest "relative error" lower bound, for the Traveling Salesman Problem is one.

We look into the assimilation of optimal algorithms with exponential time bounds to a supervised machine learning problem and show that the approximate solutions can be almost as good and much less computationally expensive. The success of this phase is attributed to geometric deep learning, which have shown promise in capturing symmetries in graph-represented knowledge and its underlying correlations.

Moreover, we emphasize on the essential connection between game theory and mathematical optimization by investigating the correspondence between optimal strategies in zero-sum games and optimal solutions in the assignment problem. This part of the research serves to chip off the limitations imposed by the supervised approach. In short, we suggest to formulate the Vehicle Routing Problem as a zero-sum game where a Nash equilibrium of the strategies played by two opposing reinforcement learning agents is approximated.

1.1 Outline

- chapter 2 builds necessary intuition and high-level understanding of mathematical optimization and the internals of a few concepts used in our work as well as discussing other relevant contributions.
  - section 2.1 explains the internals of the following topics: integer programs, relaxation in integer programs, primal-dual method, branch and bound, strong branching strategy, pseudo-reliable cost branching strategy, formulations of vehicle routing and bin packing problems as integer programs.
  - section 2.2 examines prior research on vehicle routing problem, highlighting current approaches to solving combinatorial optimization problems using one or more of end-to-end deep learning, handcrafted heuristics, or imitation learning and their limitations.
- chapter 3 introduces the formulation of the branch and bound algorithm as an imitation learning problem which is tailored to solving the Vehicle Routing Problem.
  - section 3.1 provides an overview of the method and make some remarks
  - section 3.2 details the analytical approach used in our research, including the geometric models employed and the modifications applied to them.
  - section 3.6 outlines the experimental design and introduces the equations used to inspect various performance aspects.
  - The results of the study are presented in section 3.7, including key findings and visualizations to aid in tracing patterns, correlations, and gains.
  - In section 3.8, we discuss these results and suggest potential avenues for future research.
- chapter 4 summarizes the contribution and the findings of the study.

1.2 Summary of Research Contributions

- The research contributes two novel implementations of GraphSAGE and graph attention networks for combinatorial problems.
- Additionally, we examine the performance of [1] and the newly introduced architectures on both vehicle routing and bin packing problems using a total of eight instances for training and 13 instances for evaluation;
- Eventually, we assert or refute correlations between the graph topology, running time, solution quality, number of explored nodes, and training sample size.

1.2.1 Open-source Contributions

The source code, trained models weights, and training datasets used in our research are publicly available at:

- https://github.com/isotlaboratory/ml4vrp

The implementation of the integer programs for vehicle routing and bin packing problems are pull-requested for merging to the source code of École:

- https://github.com/ds4dm/ecole/pull/352

Moreover, the same pull-request adds a new reward function that returns an estimation of the branch and bound tree size, which can be used to train a reinforcement learning agent.

Part I

Foundations

Chapter 2

Preliminaries and Literature Review

> In mathematics you don't understand things. You just get used to them.
>
> *John von Neumann*

2.1 Background

2.1.1 Integer Program (IP)

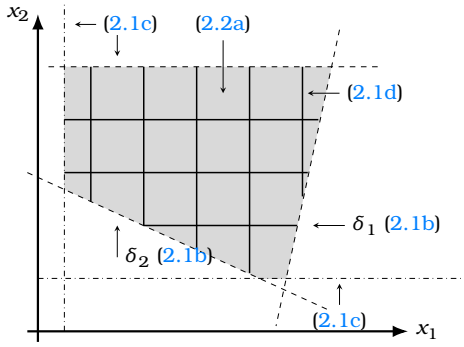


Figure 2.1: An IP example consisting of two variables and three constraints.

Let $x, c, l, u \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $I \subseteq \{1, 2, \cdots, n\}$, and $l, u \in \mathbb{R}^n$. An IP can be defined as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x && \text{(2.1a)} \\ \text{subject to} \quad & Ax \leq b, && \text{(2.1b)} \\ & l \leq x \leq u, && \text{(2.1c)} \\ & x_i \in \mathbb{Z} \quad i \in I. && \text{(2.1d)} \end{aligned}$$

A *feasible* assignment $x_i = z_i$ would satisfy the above constraints (2.1b)-(2.1d), and an *optimal* one $z^\star$, called the *minimizer*, is both feasible and minimizes the objective (2.1a). For a shorthand notation, let δ_i refer to $A_i x_i \leq b_i$ in (2.1b).

The *linear programming relaxation* (LP-relaxation) for the IP defined above would be:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x && \text{(2.2a)} \\ \text{subject to} \quad & Ax \leq b, && \text{(2.2b)} \\ & l \leq x \leq u. && \text{(2.2c)} \end{aligned}$$

That is, the integrality constraints are *relaxed*, which also makes the problem convex. Clearly, the optimal solution to (2.2a) is also optimal to the original problem (2.1a) if and only if it satisfies (2.1d).

2.1.2 Primal-Dual Method

The quality of each feasible solution found to (2.1a) is assessed using the primal-dual method. The *primal (lower) bounds* $p^\star$ are provided by feasible solutions, and the *dual (upper) bounds* $d^\star$ by *relaxation* or duality. The Lagrangian dual problem [2] to (2.2a) is:

$$\begin{aligned} \max_{\lambda} \quad & -b^T \lambda && \text{(2.3a)} \\ \text{s.t.} \quad & A^T \lambda + c \geq 0, && \text{(2.3b)} \end{aligned}$$

where λ is the Lagrange multiplier or the dual variable for the inequality constraints (2.2b). The dual problem acts as a certificate on the limit of the performance, i.e., the upper bound that declares optimality of $p^\star$. The duality *gap* f is given by $p^\star - d^\star$. A gap $f > 0$ or $f = 0$ indicates a *weak-* or *strong-duality*, respectively. We define the relative gap to be:

$$\Delta f = \frac{|p^\star - d^\star|}{min\{|p^\star|, |d^\star|\}}.$$

In the following sections, we shall drop the term "relative" and refer to Δf as gap.

2.1.3 Branch and Bound

The two building blocks in the algorithm are:

- Branching: breaking the problem into several smaller and less constrained ones;
- Bounding: selecting which subproblems to solve next.

Formally, for a set of variables, $\{x_1, x_2, \cdots, x_k\}$, the algorithm initializes the list of branching candidates $S = \{x_1\}$ (i.e., the first node), the optimal assignments $z_i^\star = \phi$, and the optimal gap $f^\star = -\infty$ (or any other heuristically known and feasible lower bound).

Then, (1) an LP-relaxation, LP_i (2.2a), is solved for the candidates in S *(bounding)*, after which they are removed. (2) If the LP_i yields an infeasible solution $z_i \notin [l_i, u_i]$ and $S = \{\}$ (i.e., no more branching candidates), the algorithm halts. Otherwise, (3) another feasibility check is made to the *original* integer program (2.1a)-(2.1d), if $z_i \in \mathbb{Z}$ (2.1d),

then $z_i^\star = z_i$ and $f^\star = f_{\text{LP}_i}$, which would then be an optimal solution. If $z_i \notin \mathbb{Z}$, (4) then sub linear programs $\text{LP}_{i0}, \cdots, \text{LP}_{in}$ are constructed from the current LP_i *(branching)* and whose union of the feasible-solution space does not contain z_i.

There are several strategies to drive the behavior of the branching and bounding steps [3, 4]. A lower bound has not yet been proven for the algorithm and still an open question [5]. The time required to solve a problem increases *exponentially* with the number of variables.

2.1.4 Strong Branching Policy (SB) and Reliable Pseudo-Cost (RPC)

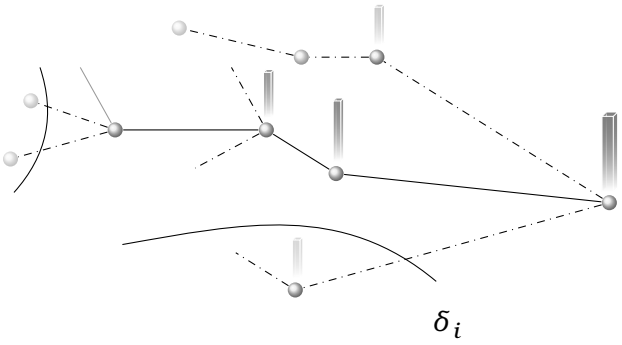


Figure 2.2: The arcs on the dashed branches are equivalent to the δ_i, The bars hovering over each node is a *visual* representation of the upper bound.

The crux of SB is to carefully branch the tree to guarantee the smallest B&B tree size by performing a one-step look-ahead before deciding to *branch*. The solver starts by choosing a set of integer variables, S, that are fractional in the LP-relaxation (when S represents all of the integer variables, it is the *Full Strong Branching* policy). Then, a temporal lower and upper bounds for the selected variable are calculated and based on which the subsequent paths are decided. A visual representation of the process is depicted in Figure 2.2, where each spherical node can be thought of as a variable and the hovering bar representing its upper bound. Also, the constraints δ_i role will be to deforest that solution tree.

The core motivation for learning the decisions made by SB is that solving two LPs for each branching node is impractically costly. SB is often used in combination with other branching strategies to balance the trade-off between solution quality and the computational cost. On the other hand, RPC avoids the situation by assigning an estimated cost to each variable based on the results of the previous subproblems, and occasionally uses SB on the *unreliable* pseudo-costs, according to some preinitialized reliability parameter. RPC is only used as a baseline for comparison against the classifiers that were trained on samples from the decisions made by the SB.

2.1.5 Capacitated Vehicle Routing Problem (CVRP)

Let $G(V, \vec{A})$ be a *complete* graph consisting of the vertex set $V = \{0\} \cup N$ where $N = \{1, 2, \cdots, n\}$ and the *directed* arcs set $\vec{A} = \{(i,j) : (i,j) \in V \times V, i \neq j\}$. There is a cost c_{ij} incurred for using an arc $a_{ij} \in \vec{A}$. For simplicity, we will assume that $c_{ij} = c_{ji}$. Each

(a) (b)

Figure 2.3: A CVRP example with $n = 40$ and $k = 5$. Given the dispersed points shown in (a), CVRP asks to find the routes presented in (b).

customer $i \in N$ has a nonnegative demand q_i. The *out-degree* of a node is denoted by $\deg^+(i)$ for $i \in V$, that is, the number of arcs $\vec{a}_{ij}$ *leaving* from node i to node $j \in V \backslash \{i\}$. Similarly, the *in-degree* of a node is denoted $\deg^-(v)$. At the depot $\{0\}$, there is a fleet of vehicles of size k with identical capacities Q. The goal is to construct a tour $\vec{T} \subseteq \vec{A}$ for each vehicle where

(i) Each customer $i \in N$ is visited exactly once and by a single vehicle or tour T;

(ii) The demand served by tour $\vec{T}$ does not exceed the capacity Q;

(iii) Each tour $\vec{T}$ start and finish at the depot (to eliminate sub-tours);

(iv) The constructed tours jointly minimize the total cost and serve the total demand $\sum_{i=1}^{n} q_i$.

The corresponding integer program is:

$$\begin{aligned}
\underset{x}{\text{minimize}} \quad & \sum_{(i,j)\in\vec{A}} c_{ij}x_{ij} && && \text{(2.4a)}\\
\text{subject to} \quad & \sum_{j\in\deg^+(i)} x_{ij} = 1 && \forall\ i \in N, && \text{(2.4b)}\\
& \sum_{i\in\deg^-(i)} x_{ij} = 1 && \forall\ j \in N, && \text{(2.4c)}\\
& \sum_{j\in\deg^+(0)} x_{0j} = k, && && \text{(2.4d)}\\
& \sum_{j\in\deg^-(0)} x_{0j} = k, && && \text{(2.4e)}\\
& u_i - u_j + Qx_{ij} \le Q - q_j && \forall\ (i,j) \in \vec{T}, && \text{(2.4f)}\\
& q_i \le u_i \le Q && \forall\ i \in N, && \text{(2.4g)}\\
& x_{ij} \in \{0,1\} && \forall\ (i,j) \in \vec{A}, && \text{(2.4h)}
\end{aligned}$$

where $x \in \mathbb{Z}^{|V|\times|V|}$ and $u \in \mathbb{R}^{|\vec{T}|}$ are the decision variables representing the arc set $\vec{A}$ and a constructed tour $\vec{T}$. Both (2.4b) and (2.4c) ensure (i). The *sub-tour elimination and capacity constraints (SEC)* (ii) and (iii) are guaranteed by (2.4f) and (2.4g), which are the Miller-Tucker-Zemlin (MTZ)-formulation of SEC [6]. The *integrality constraints* are enforced by (2.4h).

2.1.6 Bin Packing Problem (BPP)

A formulation of the IP for BPP in terms of CVRP can be written by removing constraints (2.4f)-(2.4g) and through a simple algebraic manipulation to minimize the number of vehicles, referred to as bins in this context, in lieu of (2.4a). This is achieved by expanding the value k to the *bins* variable $K = \{1, 2, \cdots, k\}$, and conditioning that the demands q_j for $j \in N$, represented as weighted items, are all distributed across the available bins without exceeding the bin capacity Q.

$$\begin{aligned}
\underset{k}{\text{minimize}} \quad & \sum_{i}^{k} k_i && && \text{(2.5a)}\\
\text{subject to} \quad & \sum_{i}^{k} q_j x_{ij} \leq Q k_i && \forall\, j \in N, && \text{(2.5b)}\\
& \sum_{i}^{k} x_{ij} = 1 && \forall\, j \in N, && \text{(2.5c)}\\
& k_i \in \{0, 1\} && \forall\ i \in K, && \text{(2.5d)}\\
& x_{ij} \in \{0, 1\} && \forall\ i \in K, j \in N. && \text{(2.5e)}
\end{aligned}$$

The decision variable x represents the allocation of customers' demands or items to vehicles or bins, with each entry $x_{i,j}$ representing the assignment of the i-th vehicle or bin to the j-th demand or item.

We reformat equation (2.5b) as $\sum_i^k q_j x_{ij} - Q k_i \leq 0$ to match the format required by a SCIP constraint. Using this integer program, we can determine the value of $k_{\min}$, which is the minimum between the optimal solution for (2.5a) and the strict lower bound obtained by $\lceil \frac{\sum_i^n q_i}{Q} \rceil$. Equation (2.5b) represents the bin capacity constraint, (2.5c) ensures that each item is assigned to a single bin, and (2.5d)-(2.5e) are the integrality constraints.

2.2 Literature Review

Solving optimization problems using neural networks emerged from the work of [7], who formulated the TSP as an energy minimization problem and showed that the energy function decreases as the algorithm (a neural network) progresses to a local optimum; that said, it was still behind several heuristics. Attempts to utilize the same idea for CVRP were first examined by [8, 9]. Although the internals of the two approaches differs, they both employ self-organizing feature maps and show on-par performance with other heuristics such as [10, 11]. Building a search tree based on the Branch and Bound remained a promising approach for decades and many have contributed guiding heuristics and branching policies.

The effectiveness of using a machine learning model to imitate the decisions of the B&B and replace the underlying branching policy was first studied by [12]. They trained a surrogate function to rank the decisions collected from the SB branching policy, treating it as a learning-to-rank problem. Their results showed promise for this approach. Later, [1] examined using deep neural networks to further improve this idea, formulating the

B&B as a Markov Decision Process and evaluating it on four $\mathcal{NP}$-hard problems: set cover, capacitated facility location, combinatorial auction, and maximum independent set. Notably, [13] were able to build on this work and develop a batch LP-solver for IP that generates 2.6 times more data samples than its sequential counterpart, using the Alternating Direction Method of Multipliers [14], which breaks an optimization problem into sum of two or more simpler functions.

Another method was put forward by [15], who handcrafted input features with an improved signal-to-noise ratio by adding a richer context of the variable selection process (for instance, adding the variable participation history in the search process and previous branchings). The authors also contributed two novel deep neural network architectures and empirically showed that the GCNN framework proposed by [1] poorly generalizes to generic MIP problems compared to theirs. Since modern solvers are inherently CPU-based, [16] investigated the possibility of using GNN (which is GPU-intensive) only at the B&B root node, where the rest of the tree is explored using the simple Multi-Layer Perceptron.

End-to-end learning for optimization problems started only seven years ago when [17] introduced the Pointer Network, which provided sub-optimal solutions to three geometric tasks, one of which is TSP. The authors start by solving the problem (up to 20 nodes) using the Held-Karp algorithm [18] whose time complexity is $O(2^n n^2)$, and generating a labeled training set (solving larger instances was non-viable due to high computational costs). The approach was extended by [19] to be of a reinforcement learning nature by predicting the distribution of possible routes permutations (similar to having a value function) and guiding the agent with the negative tour length as the reward signal.

For CVRP, [20] used the original Pointer Networks without the encoding step, which uses a recurrent neural network, arguing that it adds a layer of complexity (since there is no sequential information in TSP) and that the model generalizes to larger problems without it. Learning heuristic methods was also investigated by [21], who proposed an attention-based encoder-decoder architecture trained using REINFORCE with baseline [22], their architecture expects graph nodes (e.g., customers, cities, etc.) as input and outputs a permutation of those nodes as the output (the optimal tour found). Despite being designed for TSP, one can modify the architecture to work on other problems.

For an overarching study of machine learning approaches for combinatorial optimization, we refer the reader to [23]. In addition, [24, 25] can be regarded as the handbook of CVRP and BPP, respectively.

Our research applies the theoretical framework of [1, 12, 13] to solving CVRP, including providing an estimate of the minimum number of required vehicles by additionally solving the BPP. Moreover, we scrutinized the generalization error of such an approach concerning those two problems. We conducted a rich empirical analysis that included numerous topological structures (i.e., different dispersions of customers' locations) to confirm similarly learned patterns and refute incongruous ones.

Part II

Analysis

Chapter 3

Geometrical Models for Combinatorial Optimization

> If a neural network was invariant to some group, then its output could be expressed as functions of the orbits of the group
>
> Group Invariance Theorem

3.1 The Routing Problem

Optimum routing of a fleet of vehicles, referred to as *capacitated vehicle routing problem (CVRP)* is an *intractable* task that has been recognized since its first highlighting by [26] under the name "Truck Dispatching Problem". Simply put, given a fleet of vehicles based at a depot and a set of customers' demands, the task is to determine a tour for each vehicle such that they jointly minimize some cost (e.g., total distance traveled) while satisfying a set of constraints (e.g., demands served per tour must not exceed the vehicle's capacity). When the fleet size is one, the problem reduces to the traveling salesman problem.

CVRP is closely related to the *bin packing problem (BPP)*. In fact, when the cost of traveling between customers is zero, CVRP is equivalent to BPP. Additionally, the minimum number of vehicles required to produce valid solutions in a CVRP instance can be computed by reformulating it to BPP, which is our focus here. Generally, BPP call for distributing a set of items across a finite number of bins with limited capacities, in a way that the total weight of all the items in each bin is less than its capacity.

Both CVRP and BPP are NP-hard problems [25, 27]. Exact solutions (or approximate ones [28, 29]) cannot be computed *efficiently* unless $P = NP$. Therefore, heuristics with bounded-error ratios are often used [30], but these are not general-purpose strategies and cannot be relied upon in all cases [31]. In this work, we contribute multiple statistical methods that are particularly specialized for CVRP and BPP.

As a combinatorial optimization problem, CVRP or BPP can be formulated in a number

of ways, our stance here will be that of an integer program (2.1.1). There are three classes of questions relevant to designing statistical methods for integer programs about which we would like to make some remarks: (A) how can geometric deep neural networks [32] be used to approximate solutions to CVRP and BPP as integer programs? (B) what is the trade-off between solution quality, running time, and sample complexity when using this approach? (C) how can we assess the generalization error of these methods collectively and across CVRP graph topologies different from those the geometric neural networks were trained to solve?

The central component our work is to assimilate the branch and bound (B&B) [33] decisions when solving CVRP and BPP to a supervised learning problem. B&B is a powerful algorithm for solving integer programs using tree-based solution search space. The success of the algorithm comes from the impressive strategies for pruning infeasible solutions paths down the tree, two of which are strong branching (SB) and reliable pseudo-cost (RPC) [34, 35] and the strategies we use. Although SB being computationally expensive, it guarantees to obtain the optimal solution in the smallest B&B tree size (the internals of B&B, SB, and RPC are explained in Sections 2.1.3 and 2.1.4).

For problem (A), we use the graph convolutional neural networks (GCNN) by [1], and design two others *based on* graph attention network [36] and GraphSAGE [37]. We train each of these methods on decision samples from B&B when solving CVRP on instances from CVRPLIB [38] and BPP on instances from the Operation Research Library [39], which were generated using a modified version of École [40]. Under the hood, École uses SCIP [41] as the underlying mathematical solver.

(B) Next, we evaluated the solution quality of SB and RPC as the branching strategy against each of the resulting classifiers (which at this point are considered strategies as well) from (A) . Our goal was not to outperform SB (although the results show that this occurred in many experiments), but rather to provide a gain measure representing the compensation for the relative primal-dual gap (2.1.2) in terms of the running time. Through eight hundred experiments, we evaluate the competence of the classifiers, and show that they are consistently able to find solutions that are the same as, sometimes better than, or within a small fraction of the accuracy of the solution found by the SB or RPC in 2x-8x *less* time using 10^5 decision samples.

The challenge in (C) is defining what constitutes the difficulty of an integer program, which has the following contributing factors: the instance topology, the nature of the constraints, cardinality of the input, and the number of binary variables in the integer program. All together, these factors also impact the tree-size estimate (TSE) of the B&B [42–44], which we report along with the graph edit distance (GED) [45] for each training and evaluation instances pair, in an attempt to yield a credible relationship between the solution quality and the problem complexity.

The rest of the chapter is structured as follows: In Section 2.1 , we present an overview of preliminary concepts and mathematical formulations used in our work. Section 2.2 examines prior research on CVRP, highlighting current approaches and limitations. Section

$$
\begin{aligned}
\min_{x} \quad & c_1 x_1 \quad + c_2 x_2 \quad + \cdots + c_n x_n \\
\text{s.t.} \quad & a_{11} x_1 \; + a_{12} x_2 + \cdots + a_{1m} x_n \leq \; b_1 \\
& \vdots \qquad\qquad\qquad \ddots \quad \vdots \qquad\quad \vdots \\
& a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n \leq \; b_m
\end{aligned}
$$

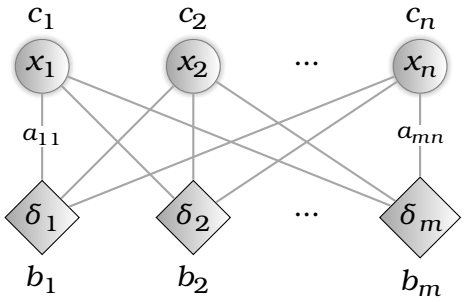


Figure 3.1: portraying equations (2.1a)-(2.1b) as a bipartite graph; the corresponding integer program would then be $\min_{\circ} \sum \circ$; s.t. $\diamond$

3.2 Conceptualization as a Supervised Learning

We model each branching step, as described in Section 2.1.3, using a bipartite node with two disjoint sets, denoted as $x = \{x_1, x_2, \cdots, x_n\}$ and $\delta = \{\delta_1, \delta_2, \cdots, \delta_m\}$ (see Figure 3.1), representing the variables and constraints in (2.1b), respectively. The sensitivity of a variable to the objective function (either (2.4a) for CVRP or (2.5a) for BPP) is represented by an edge weight, or coefficient, $a_{ij} \in A$, where i belongs to set x and j belongs to set δ. Each element in either sets has a feature vector that describes its attributes during the solving process. For example, the feature vector for a variable $i \in x$ may include the objective value, variable type (such as binary, integer, or continuous), and lower and upper bounds. Similarly, the feature vector for a constraint $j \in \delta$ may include the tightness, dual problem solution value (2.3a), scaled age, and bias. In total, there are 19 features collected for each variable and 5 for each constraint.

The three graph neural networks—GCNN, GraphSAGE, and GAT—will be described using the message-passing protocol [46]. Although the networks may differ in their implementation of the AGGREGATE and UPDATE operations, they all operate on the same basic principle. These operations, which are differentiable functions, are responsible for aggregating information from a node's local neighborhood and updating the node's features according to the aggregated *message*.

3.3 Graph Convolutional Neural Network (GCNN)

In the GCNN designed by [1], the input is represented by a triplet (δ, A, x). The first layer transforms the elements to the latent vectors $H_\delta \in \mathbb{R}^{m \times d}$, $H_A \in \mathbb{R}^{m \times n \times d}$, and $H_x \in \mathbb{R}^{n \times d}$, where d is the embedding dimension.

For both δ and x, the transformation includes layer normalization (LayerNorm, as

proposed by [47]), a linear transformation, and a rectified linear unit (ReLU) as the non-linearity, which we denote as σ. For convenience, we construct f to perform the embedding of the elements $i \in H_x$ and $j \in H_\delta$ as follows:

$$f(y) = \begin{cases} y, & \text{if } l = 0 \\ \sigma\left(W^{(0)}\text{LayerNorm}(y)\right), & \text{otherwise} \end{cases} \tag{3.1}$$

$$H_x^{(0)} = \{\underbrace{f(x_1)}_{h_{x_1}^{(0)}}, \underbrace{f(x_2)}_{h_{x_2}^{(0)}}, \cdots, \underbrace{f(x_n)}_{h_{x_n}^{(0)}}\}, \tag{3.2}$$

$$H_\delta^{(0)} = \{\underbrace{f(\delta_1)}_{h_{\delta_1}^{(0)}}, \underbrace{f(\delta_2)}_{h_{\delta_2}^{(0)}}, \cdots, \underbrace{f(\delta_m)}_{h_{\delta_m}^{(0)}}\}, \tag{3.3}$$

where $W^{(l)}$ is a weights matrix at layer (l), and is distinctive among the expressions. For the transformation of A, only the LayerNorm is used:

$$H_A^{(0)} = \underbrace{\text{LayerNorm}\left(\mathbf{A}_{ij}\right)}_{h_{A_{ij}}}, \qquad \forall\ (i,j) \in A. \tag{3.4}$$

The authors apply a custom graph convolution to the produced latent vectors in two phases. In the first phase, they convolve over the triplet (H_x, H_A^T, H_δ) to compute the message $m_{(ij)}^{(l)}$, which is the convolution on the constraints side, let it be $\mathscr{C}_{x\text{-}\delta}$. This convolution is calculated as follows:

$$\begin{aligned} h_{ij}^{(l)} &= W^{(l)} h_{x_i}^{(l-1)} + W^{(l)} h_{A_{ij}}^{T\,(l-1)} + W^{(l)} h_{\delta_j}^{(l-1)}, \\ \bar{h}_{ij}^{(l)} &= \text{LayerNorm}\left(W^{(l)}\left(\sigma\left(W^{(l)} h_{ij}^{(l)}\right)\right)\right). \end{aligned}$$

The essence of this step is to learn the effect of each variable on the constrained region. The AGGREGATE operation is defined as reduction by summation of all normalized features from the neighboring nodes in the set δ, which is referred to as message m_{ij}:

$$\begin{aligned} \text{AGGREGATE}^{(l)}(H_x, H_A, H_\delta) &= \underbrace{\sum_{j \in \mathscr{N}(i)} \bar{h}_{ij}^{(l)}}_{m_{ij}^{(l)}}, \\ \mathscr{C}_{x\text{-}\delta}^{(l)} = \text{UPDATE}^{(l)}(\{i \in |H_x|\}) &= W^{(l)} h_{x_i} + \\ & \quad W^{(l)}\left(m_{ij}^{(l)}\right). \end{aligned} \tag{3.5}$$

$\mathscr{N}(i) = \{j | A_{ij} \neq 0\ \forall\ j \in |A_i|\}$ is the set of neighboring nodes and $|\cdot|$ is the cardinality operator. In the second phase, the same operations are used but to convolve over the triplet $(C_{x\text{-}\delta}, A, x)$, this is to capture the similar concept of learning, but from the side of the constraints, $\mathscr{C}_\delta$, against the previously produced variables convolution, let it be defined as follows:

$$\begin{aligned} \mathscr{C}_{\delta\text{-}x}^{(l)} = \text{UPDATE}^{(l)}(\{i \in |H_\delta|\}) &= W^{(l)} h_{\delta_i} + \\ & \quad W^{(l)}\left(\mathscr{C}_{x\text{-}\delta}^{(l)}\right). \end{aligned} \tag{3.6}$$

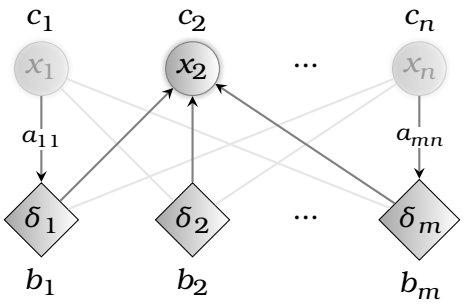


Figure 3.2: GraphSAGE accumulates information from a selection of neighboring nodes, with more distant nodes becoming increasingly influential in the aggregated information as the process continues.

Eventually, log probabilities over possible branching candidates are estimated by a non-linear transformation of the resulting convolution $\mathscr{C}_{\delta\text{-}x}$, the branching candidate is the one with highest log probability:

$$\hat{v} = \operatorname{argmax}\left(W^{(l+1)}\left(\sigma\left(W^{(l+1)}\mathscr{C}^{(l)}_{\delta\text{-}x}\right)\right)\right). \tag{3.7}$$

3.4 GraphSAGE

The inputs to non-bipartite geometric classifiers are curated differently; the constraints $\delta_{n\times 5}$ are first padded with zeros to match the variables row dimension, resulting in $\delta^{pad}_{m\times 19} = [\delta_{m\times 5}\ \mathbf{0}_{m\times 14}]$. Then, the variables $x_{n\times 19}$ and the padded constrains $\delta^{pad}_{m\times 19}$ are augmented to form the input $H = \begin{bmatrix} x_{n\times 19} \\ \delta^{pad}_{m\times 19} \end{bmatrix} \in \mathbb{R}^{(n+m)\times 19}$, which is then fed en bloc to the classifier. The adjacency matrix would then be a zero-diagonal matrix with the coefficients being off-diagonal[1]:

$$\mathbf{A} = \begin{array}{c} \\ n \\ \\ m \\ \\ \end{array} \overbrace{\begin{array}{c} \\ \\ \\ \\ \\ \end{array}}^{n} \overbrace{\begin{array}{c} \\ \\ \\ \\ \\ \end{array}}^{m} \begin{bmatrix} 0 & \cdots & 0 & a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n1} & \cdots & a_{nm} \\ a_{11} & \cdots & a_{1n} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} & 0 & \cdots & 0 \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{0} & A \\ A^T & \mathbf{0} \end{bmatrix}.$$

[1]The matrix can be constructed and stored efficiently using coordinate format [48].

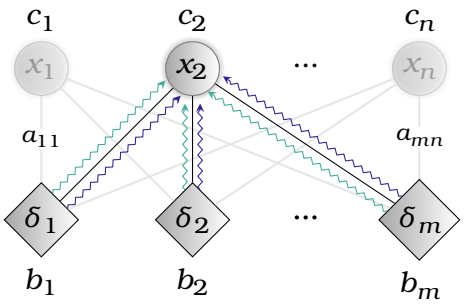


Figure 3.3: GAT assigns attention scores to the relationships between each node and its neighboring ones. This is an example of a three-head attention mechanism.

We start by taking a linear transformation of the inputs to produce $H' \in \mathbb{R}^{(n+m)\times 19\times d}$:

$$H'^{(0)} = \left[\underbrace{W^{(0)}x_1}_{h'^{(0)}_1}, \cdots, \underbrace{W^{(0)}x_n}_{h'^{(0)}_n}, \underbrace{W^{(0)}\delta_1}_{h'^{(0)}_{n+1}}, \cdots, \underbrace{W^{(0)}\delta_m}_{h'^{(0)}_{n+m}} \right], \tag{3.8}$$

where $W^{(l)} \in \mathbb{R}^{d\times 19}$. The message produced for node $i \in H'$, denoted m_i is defined as:

$$\begin{aligned} m_i^{(l)} &= \text{AGGREGATE}^{(l)}(h_i) = \underbrace{\frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} h_j^{(l)}}_{\text{Neighborhood Averaging}}, \\ \text{UPDATE}^{(l)}(\{\forall\ i \in |H'^{(l)}|\}) &= \sigma\begin{pmatrix} W^{(l)}h_i + \\ W^{(l)}m_i^{(l-1)} \end{pmatrix}, \end{aligned} \tag{3.9}$$

where $\mathcal{N}(i) = \{j\ |A_{ij} \neq 0, \forall j \neq i \in |H'|\}$. Log probabilities over the branching candidates are obtained by a linear transformation over the updated features. The branching candidate is then selected using:

$$\hat{v} = \text{argmax}\left(W^{(l+1)}H'^{(l)}\right) \tag{3.10}$$

3.5 Graph Attention Network (GAT)

The input to the GAT is the same as GraphSAGE, H. Similarly, H' is calculated according to (3.8). We then learn a self-attention mechanism $a : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, defined as:

$$a^{(l)}(h'_i, h'_j) = \text{LeakyReLU}\left(w_a^{T(l)} \odot \left[h'_i \| h'_j\right]\right),$$

where $\text{LeakyReLU}(y) = \max(0, y) + m * \min(0, y)$, $w_a^{T(l)} \in \mathbb{R}^{2d}$, and $\|$ denotes the concatenation operator. The attention score a_{ij} is then generated by normalizing the attention values assigned to the neighboring nodes using a through the use of the softmax function and multiplying by $\mathbf{A}_{ij}$ in the following way:

$$a_{ij}^{(l)} = \frac{\exp\left(a^{(l)}(h_i, h_j)\right)}{\sum_{j\in\mathcal{N}(i)}\exp\left(a^{(l)}(h_i, h_j)\right)} \odot \mathbf{A}_{ij}.$$

Such score indicates the learned connectivity strength of node i to node j rescaled by the decision variable coefficient. The AGGREGATE operation takes a non-linear combination of the learned attention scores and the linearly transformed inputs:

$$\text{AGGREGATE}^{(l)}(i) = \sigma\left(\sum_{j\in\mathcal{N}(i)} a_{ij}h'_j\right).$$

The non-linearity, σ, is used to avoid unintended fluctuations in the learned attention values. To integrate a K-head attention mechanism, for $K \in \mathbb{Z}^+$ (see Figure 3.3), the outputs are averaged among K independent operators, the message-protocol is constructed by:

$$\begin{aligned} m_i^{(l)} &= \frac{1}{K_{(l)}}\left(\sum_{k=1}^{K_{(l)}} \text{AGGREGATE}^{(l)}\ (h_i)\right) \\ \text{UPDATE}^{(l)}(\{\forall\ i \in |H'^{(l)}|\}) &= \sigma\begin{pmatrix} W^{(l)}h'_i + \\ W^{(l)}m^{(l-1)}(i) \end{pmatrix}. \end{aligned} \tag{3.11}$$

The branching candidate is decided, analogously to GraphSAGE, by:

$$\hat{v} = \text{argmax}\left(W^{(l+1)}H'^{(l)}\right) \tag{3.12}$$

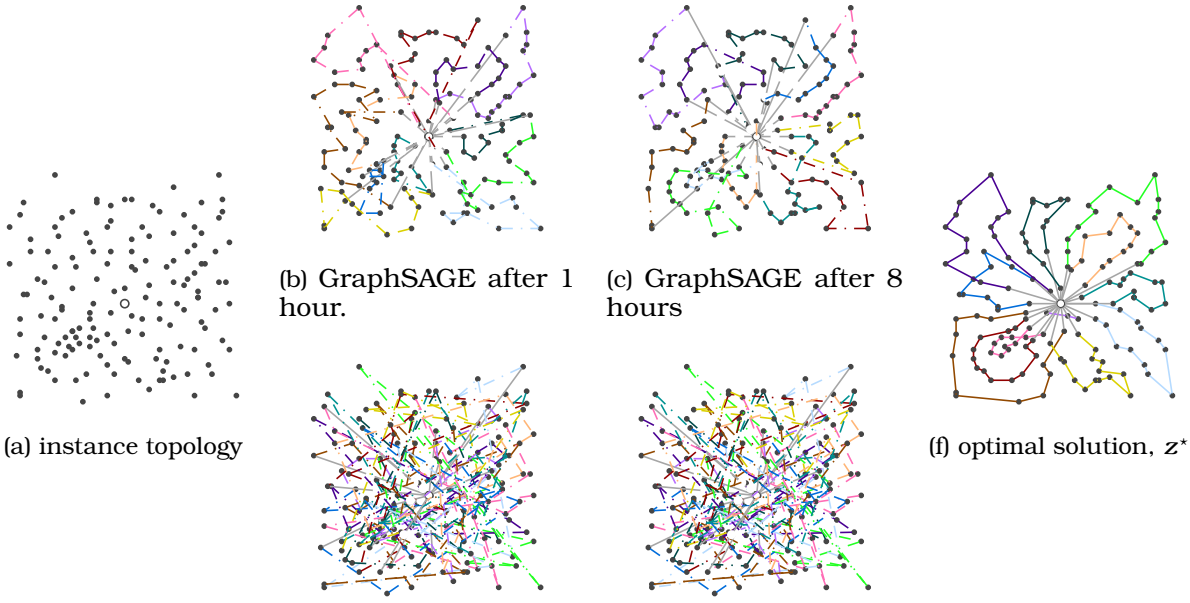
3.6 Experiments

3.6.1 Datasets

The SB decision samples were drawn using École for the CVRP and BPP integer programs, as outlined in sections 2.1.5 and 2.1.6. The CVRP training and evaluation instances were drawn from the 12 benchmark sets in CVRPLIB [38], with six instances from sets A and P [49] for training (`A-n32-k5`, A-n33-k5, A-33-k6, A-n39-k5, A-n44-k6, and P-n40-k5) and eight instances from sets P, B, and M for evaluation: (P-n76-k5, P-n60-k15, P-n65-k10, B-n78-k10, B-n64-k9, B-n57-k7, M-n101-k10, and M-n151-k12). Each instance is comprised of the following attributes: the number of customers, the demand of each customer, and the location of the depot and customers in a two-dimensional Euclidean space, where the cost c_{ij} in (2.4a) corresponds to the Euclidean *distance* between node i and node j.

The BPP training instances were taken from [39], specifically sets U and T, with two instances for training (u100_00 and u80_00) and five instances for evaluation (t249_00, t120_00, t60_00, u250_00, and u500_00)

The training and evaluation instances will be referred to as **tr** and **ts**, respectively. The configuring parameters in SCIP, the underlying mathematical solver used by École, were set to their default, except for disabling the separation routine, enabling the search-tree-size profiling, and the branching strategy idempotency.



(a) instance topology

(b) GraphSAGE after 1 hour.

(c) GraphSAGE after 8 hours

(f) optimal solution, $z^\star$

(d) SCIP_{SB} after 1 hour

(e) SCIP_{SB} after 8 hours

Figure 3.4: Example on solving CVRP instance M-n151-k12 using both GraphSAGE and SCIP with allowed runtime of 8 hours.

3.6.2 Training

At each branching step, we randomly set the strategy to either RPC or SB, and only the variable branchings decided by SB were exported to the training *dataset*.The cardinality N is 10^5 for each training dataset $\mathscr{D} = \{(x_i, A, \delta_i, v_i)\}_{i=1}^{N}$, where each record is an independent and identically distributed (iid) sample from the SB decisions. There are $|\mathbf{tr}|$ training datasets either for CVRP or BPP.

The classifiers were trained to maximize the log likelihood [50] using the adaptive stochastic gradient ascent [51]:

$$\mathscr{L}(\hat{v}, v) = \frac{1}{N} \sum_{i=1}^{N} \log \pi_\theta \left(\hat{v}_i = v_i | x_i, \delta_i\right) \; d\theta,$$

which can be verbally explained as: given a set of constraints δ_i, the variable features x_i, and the true branching candidate v_i, we aim to maximize the likelihood that a classifier, or policy, π parameterized by weights θ selects the same branching decision, $\hat{v}_i$, using either of (3.7), (3.10) or (3.12).

At each iteration, we tested the performance on a sequestered validation set. We did not use dropout or add self-loops. The tuning parameters of each classifier are set to the following values[2]:

- GCNN: $d = 64$, $l = 1$, $m = 0.2$;

[2]The technical details and usage instructions, including information about the other hyper-parameters that were tuned and the specific details of how to use any of the classifiers for other training and evaluation instances, are provided in the supplementary material.

- GraphSAGE: $d = 64$ and $l = 5$;
- GAT: $d = 128$, $l = 2$, $K_1 = 2$, and $K_2 = 4$.

These values were chosen by an experiential process that targeted minimizing the empirical *training* error (to avoid an overuse of the validation sets [52–54]).

3.6.3 Evaluation

We evaluated five branching strategies: SB and RPC, which are the inherently built branching strategies, and the learned ones by GCNN, GraphSAGE, and GAT using SCIP with the following time constraints: 1 hour, 2 hours, 4 hours, and 8 hours when solving the evaluation instances **ts**. For brevity, the gap achieved by SCIP using the SB or RPC strategies will be referred to as SCIP_{SB} and SCIP_{RPC}, respectively. We also use the notation SCIP_s to refer to either of them.

The optimality gain we originally alluded to in the introduction is defined using four-faceted comparison. Let $\Delta f_{j,t}^{\text{SCIP}_s}$ be the gap of SCIP using strategy s on $\mathbf{ts}_j$, the j^{th} instance of **ts**, after t hours of solving; and let $\Delta f_{j,t}^{m_i}$ be the gap of model m when trained on $\mathbf{tr}_i$, the i^{th} instance of **tr**, and evaluated on $\mathbf{ts}_j$ after t hours of solving. Then, we assess the quality of our proposed approach by defining the following relative performance measure:

$$\Delta f_{\text{SCIP}_s}^{m}(i,j,t,k) = \frac{\Delta f_{j,t}^{m_i}}{\Delta f_{j,(2^k \cdot t)}^{\text{SCIP}_s}}, \tag{3.13}$$

where we chose $k = 0, \dots, 3$, $t \in T = \{1, \dots, 2^{3-k}\}$. The measure (3.13) expresses the ratio between the gap of two solutions: (1) the proposed approach, using model m, trained on $\mathbf{tr}_i$, evaluated on instance $\mathbf{ts}_j$, and the gap is reported after t hours; (2) the solver SCIP, using the policy s, evaluated on the same instance $\mathbf{ts}_j$, and the gap is reported after $2^k t$ hours. The elongation factor 2^k, $k = 0, \dots, 3$ gives SCIP more chance to find a solution than the proposed approach, and hence illustrates the power of the latter when it finds better gaps. For more elaboration, when $k = 0$, the gap of both solutions is reported at $\{1, 2, 4, 8\}$ to provide a contrast of the classifiers performance against SCIP_s under *equal* time limits, denoted (1:1). When $k = 1$ the gap is reported at $\{1, 2, 4\}$ and $\{2, 4, 8\}$, respectively, yielding a comparison of performance with *twice* the time limit for SCIP_s, denoted (1:2). Two further comparisons are made at (1:4) and (1:8), where $k = 2$ and $k = 3$, respectively, to determine if even larger gains are achieved by increasing the time limit by *four* and *eight* times. The comparison included all instances $\mathbf{tr}_i$, $\mathbf{ts}_j$, and $s \in \{\text{SB}, \text{RPC}\}$ for both CVRP and BPP.

Moreover, we evaluate the average performance of each classifier in two ways: (1) By fixing the training instances and calculating the mean performance of *each* time-window comparison as previously described across the evaluation instances, using the following equation:

$$\Delta f_{\text{SCIP}_s}^{m}(i,t,k) = \frac{1}{|\mathbf{ts}|} \sum_{j \in \mathbf{ts}} \Delta f_{\text{SCIP}_s}^{m}(i,j,t,k). \tag{3.14}$$

(2) By averaging the performance across both the training and evaluation instances using

the following equation (i.e., reduction by i and j):

$$\Delta f^{m}_{\mathrm{SCIP}_s}(t,k) = \frac{1}{|\mathbf{tr}|}\sum_{i\in\mathbf{tr}} \Delta f^{m}_{\mathrm{SCIP}_s}(i,t,k). \tag{3.15}$$

We also designed two more measures to help in assessing the aptitude of the classifiers to generalize in performance to larger and more complex instances. These two measures are: graph edit distance (GED) [45] and tree-size estimate (TSE) as reported by SCIP. The GED is a measure of the number of edits required to make the topology of a graph $\mathbf{tr}_i$ isomorphic to the topology of graph $\mathbf{tr}_j$. The number of edits serve to provide insights on the topological similarities between an instances i and j. The TSE in SCIP is determined by calculating a growth factor that represents the relative increase in the number of nodes between two consecutive states of the tree and among all explored paths. The TSE of a classifier m trained on instance i and used as a branching strategy to solve evaluation instance j for t hours is denoted as $\mathrm{TSE}^{m_i}_{j,t}$, while the TSE of the standard branching strategy s is denoted as $\mathrm{TSE}^{\mathrm{SCIP}_s}_{j,t}$. The comparison between these two quantities aims to assess how many more nodes the learned strategy needs to explore on larger instances compared to the standard strategy s:

$$\Delta\mathrm{TSE}^{m_i}_{\mathrm{SCIP}_s} = \frac{1}{|T|}\sum_{t}\left(\frac{\mathrm{TSE}^{m_i}_{j,t}}{\mathrm{TSE}^{\mathrm{SCIP}_s}_{j,t}}\right) \tag{3.16}$$

In the next section, the displayed values of both of the GED and $\Delta\mathrm{TSE}^{m}_{\mathrm{SCIP}_s}$ are normalized to the interval $[0,1]$.

The features collected for each variable are: objective value coefficient, type of the variable (e.g., binary, integer, implicit integer, or continuous; each is represented as an independent boolean variable), lower and upper bound, variable value, variable fraction, primal-dual optimality (a boolean representing whether the decision variable has reached either the lower or the upper bounds), age (the number of times a variable has been used in the relaxed problem and had a value of 0 in the solution), incumbent value (i.e., the best solution value), average incumbent value, basis status of the variable (e.g., lower, basic, upper zero). Similarly, for each constraint, the features are: tightness (whether the constraint is at the lowest value, i.e., the left hand-side), the dual LP solution, age, and bias (a negative normalization of the constraint's left hand-side). A thorough explanation of each feature can be found in the documentation provided by [41].

3.7 Results

We present the optimality gain described by (3.13) in Figures 3.5 and 3.6 for CVRP and Figures 3.9 and 3.10 for BPP. The presentation of the elements will be explained by using examples from the CVRP figures, and convey the same theme to the ones for BPP. Each of Figures 3.5 and 3.6, which details the comparison against $\mathrm{SCIP}_{\mathrm{SB}}$ and $\mathrm{SCIP}_{\mathrm{RPC}}$, comprises six rows for the training instances and three columns for the classifiers. We will use the notation $\mathrm{Fig}(s,m,i)$ to refer to the figure describing the results of classifier m when trained on instance i and evaluated against SCIP_s on all instances $\mathbf{ts}_j$; e.g.,

Fig(RPC, GCNN, A-n32-k5) is the top and leftmost sub-figure in Figure 3.6, representing the performance gain of GCNN when trained on instance A-n32-k5 and compared against SCIP_{RPC}. Each Fig(s, m, i) has eight interior columns, one for each $\mathbf{ts}_j$, denoted Fig(s, m, i, j); each is a lump of *four* distinctive gray-level bars to *visually* differentiate the four elongation factors 2^k defined in (3.13), and each is denoted Fig(s, m, i, j, k).

For example, the first bar ▭ is Fig$(s, m, i, j, 0)$ and has four symbols − + □ ○ to represent $\Delta f^m_{\text{SCIP}_s}(i, j, t, 0)$ at $t \in T = \{1, \ldots, 2^3\}$. The second bar ▭ is Fig$(s, m, i, j, 1)$ and has three symbols + □ ○ to represent $\Delta f^m_{\text{SCIP}_s}(i, j, t, 1)$ at $t \in T = \{1, \ldots, 2^2\}$. The other two bars follow a similar pattern for the remaining fourfold (1:4) and eightfold (1:8) time limit extensions for SCIP_s.

In a limited number of experiments, SCIP_s fails to obtain a feasible solution in a certain time limit t on an evaluation instance j; we distinguish those observations by using the same symbols as described previously, however, capped by an Ω, e.g., Ω and Ω, and shifted to the top of the bar Fig(s, m, i, j, k). A single Ω spanning the entire column Fig(s, m, i, j) indicates the same meaning across the different values of k, and accordingly, t. Equivalently, we use a flipped ℧ to designate the similar behaviour for a classifier m.

Additionally, for each Fig(s, m, i), there are two stacked lines on the top: the dotted line shows the topological *complexity* of each evaluation instance relatively to the training instance, computed by the GED; the triangle-dashed line reflects the $\Delta\text{TSE}^m_{\text{SCIP}_s}$ (3.16). The pale-red area height in Fig(s, m, i, j), ▭, highlights the standard deviation of the average gains achieved by classifier m trained on instance j against SCIP_s evaluation instance j over the different values k

If a Fig(s, m, i, j) is rendered in light blue color, ▭, it indicates the highest performance gain when solving instance j using classifier m trained on instance i against SCIP_s, e.g., Fig(RPC, GCNN, A-n33-k5, P-n65-k10)

The dashed line, - - -, fixed at $\Delta f^m_{\text{SCIP}}(i, j, t, k) = 1$ in a Fig(s, m, i) separates the results into two segments. The area below the line is of particular interest because it is the region where the performance of a classifier m trained on instance i is equal to or better than that of SCIP_s.

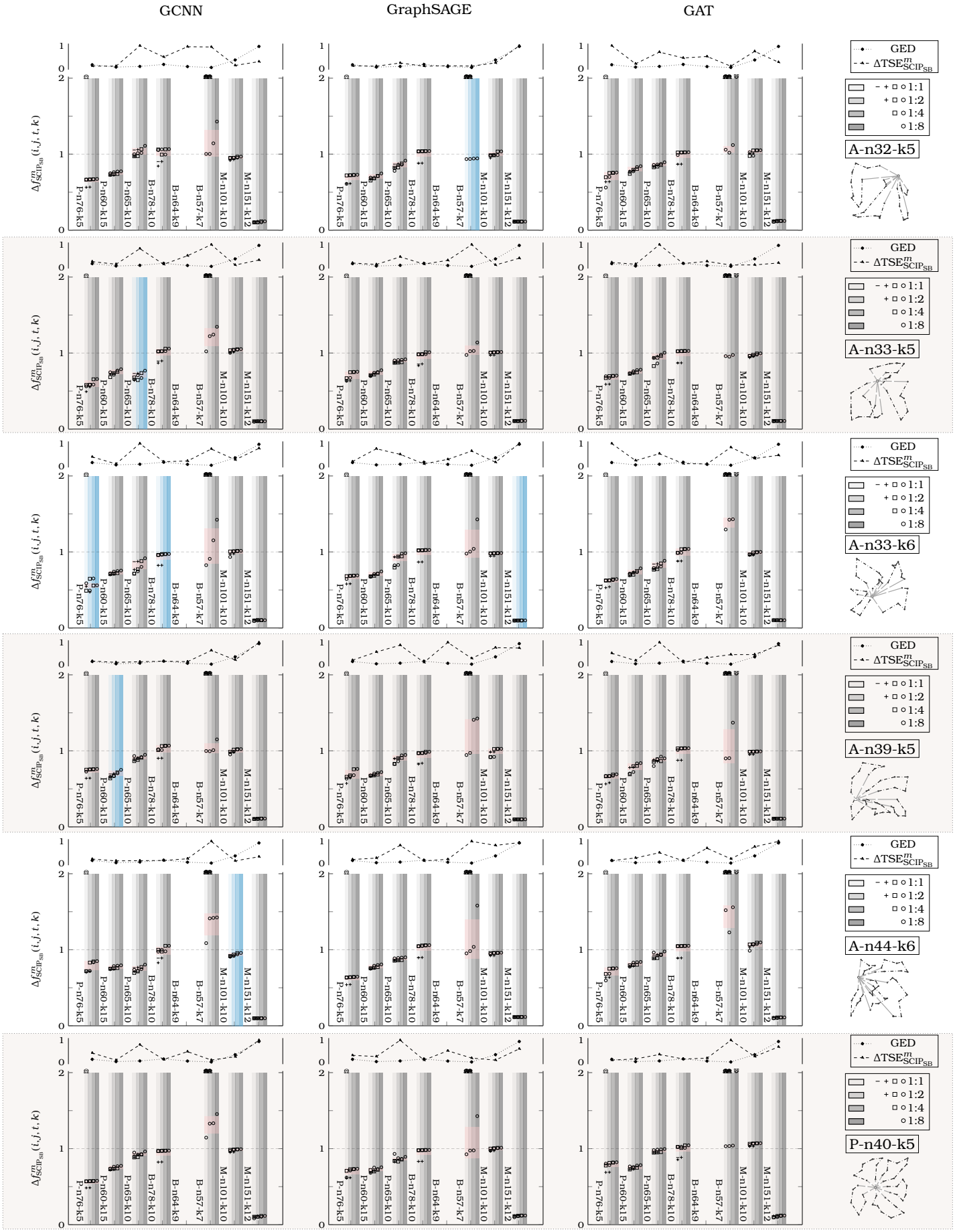


Figure 3.5: The performance of the three CVRP classifiers against $SCIP_{SB}$ calculated by (3.13), across the training instances.

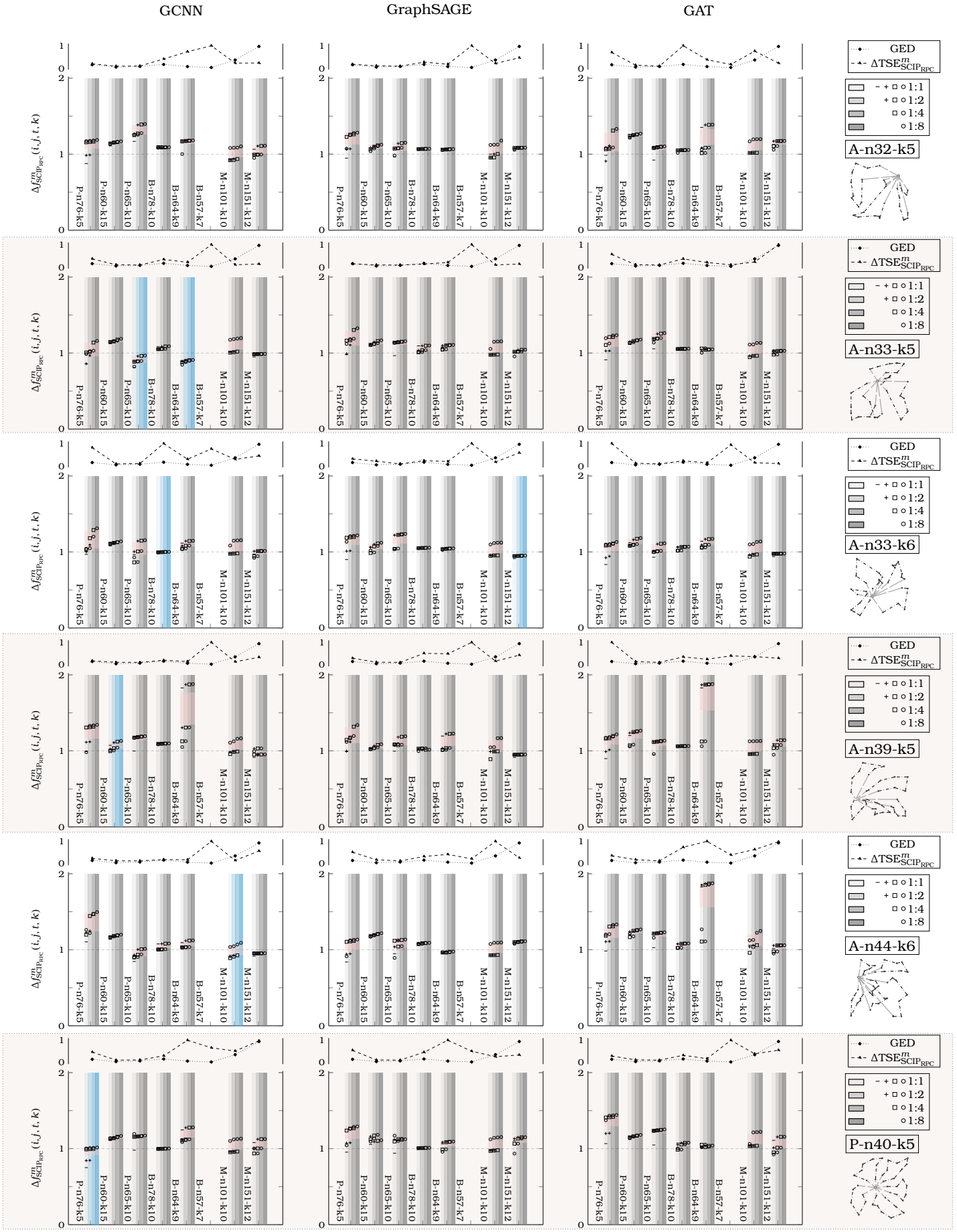


Figure 3.6: The performance of the three CVRP classifiers against $SCIP_{RPC}$ calculated by (3.13), across the training instances.

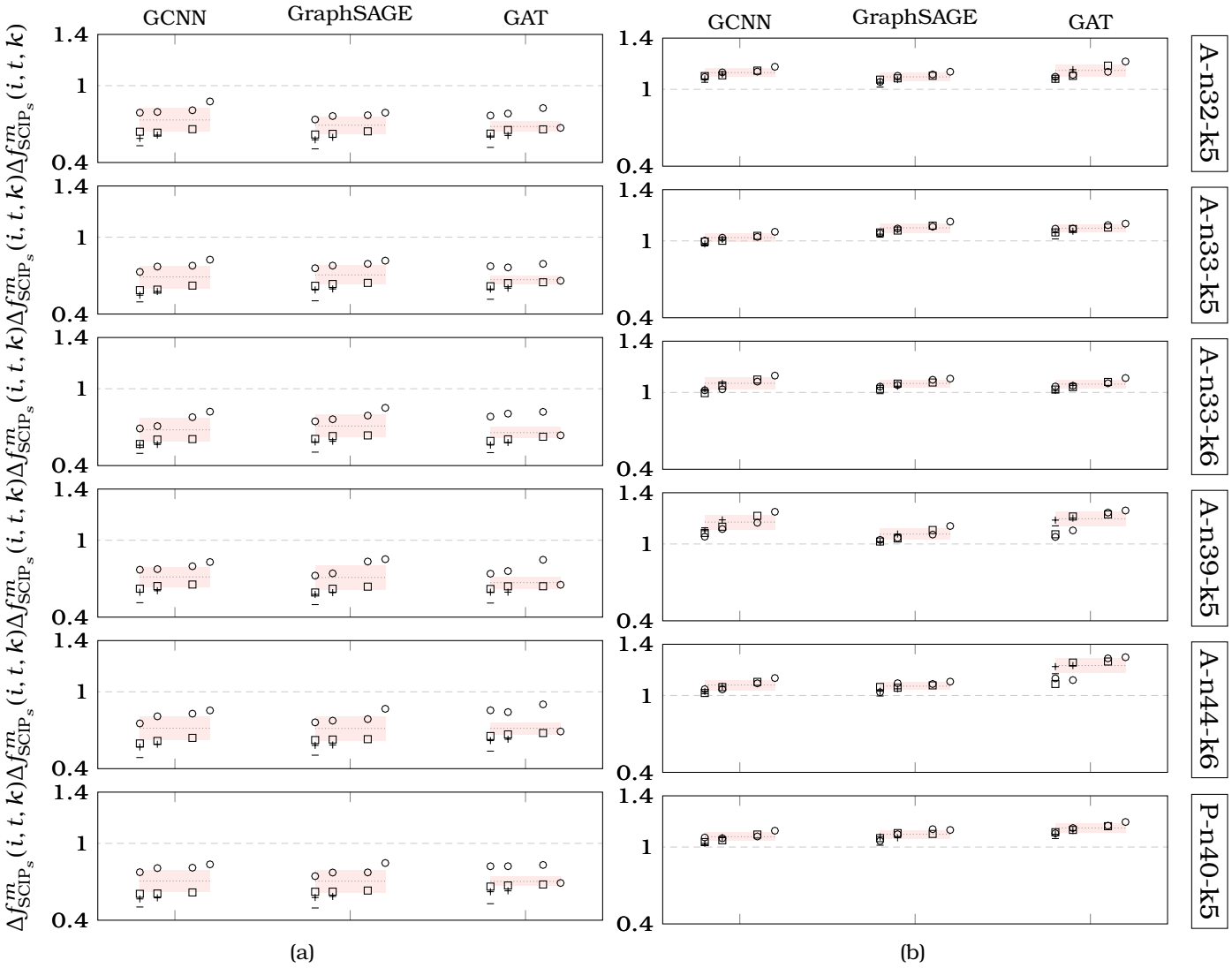


Figure 3.7: The average performance of each classifier across the CVRP evaluation instances; (a) compares against SCIP_{SB} and (b) compares against SCIP_{RPC}.

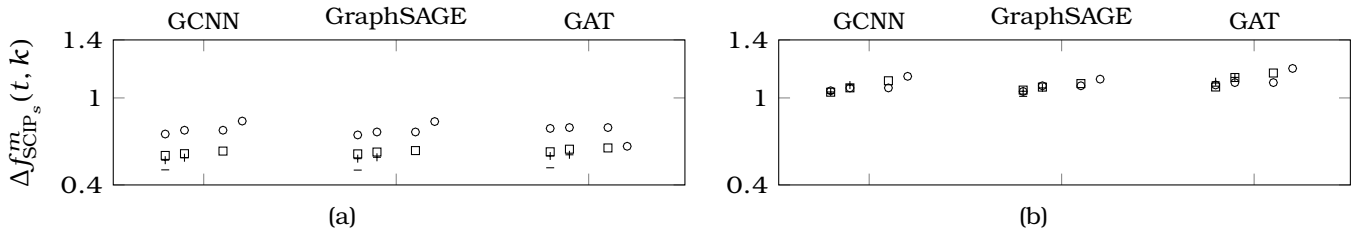


Figure 3.8: The performance of each classifier averaged by both of the CVRP training and evaluation instances against (a) SCIP_{SB} and (b) SCIP_{RPC}.

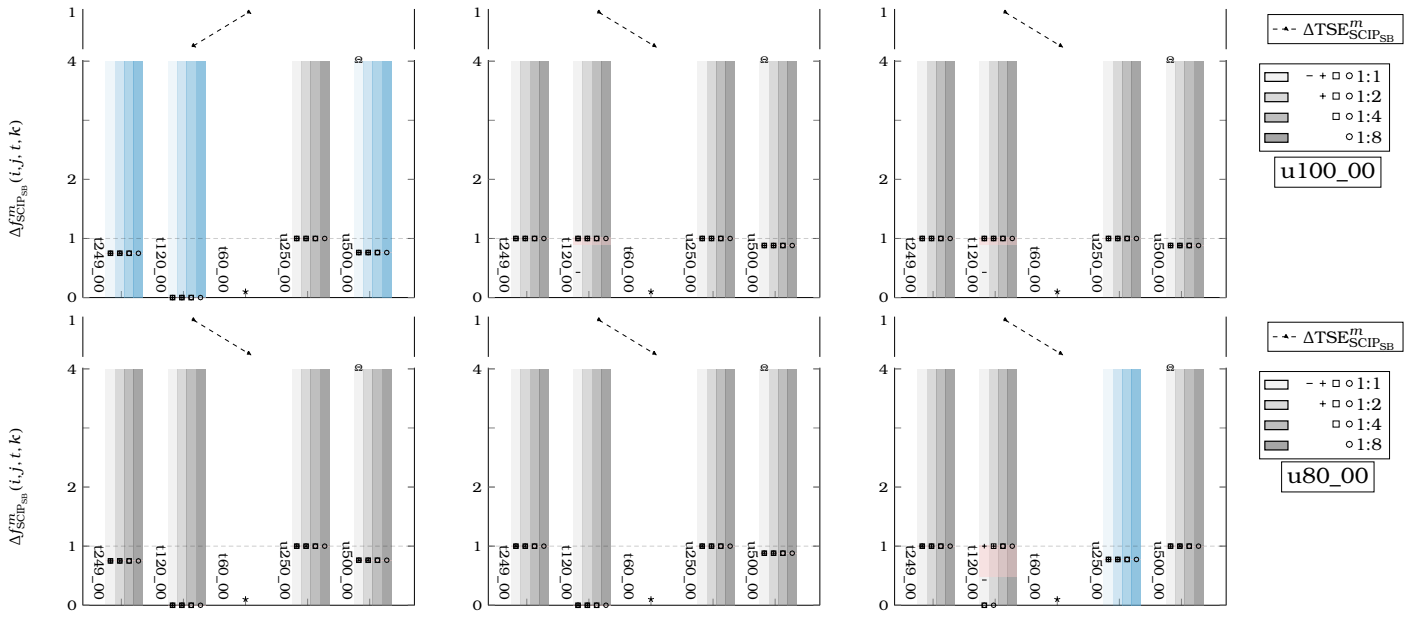


Figure 3.9: The performance of the three BPP classifiers against SCIP_{SB} calculated by (3.13), across the training instances.

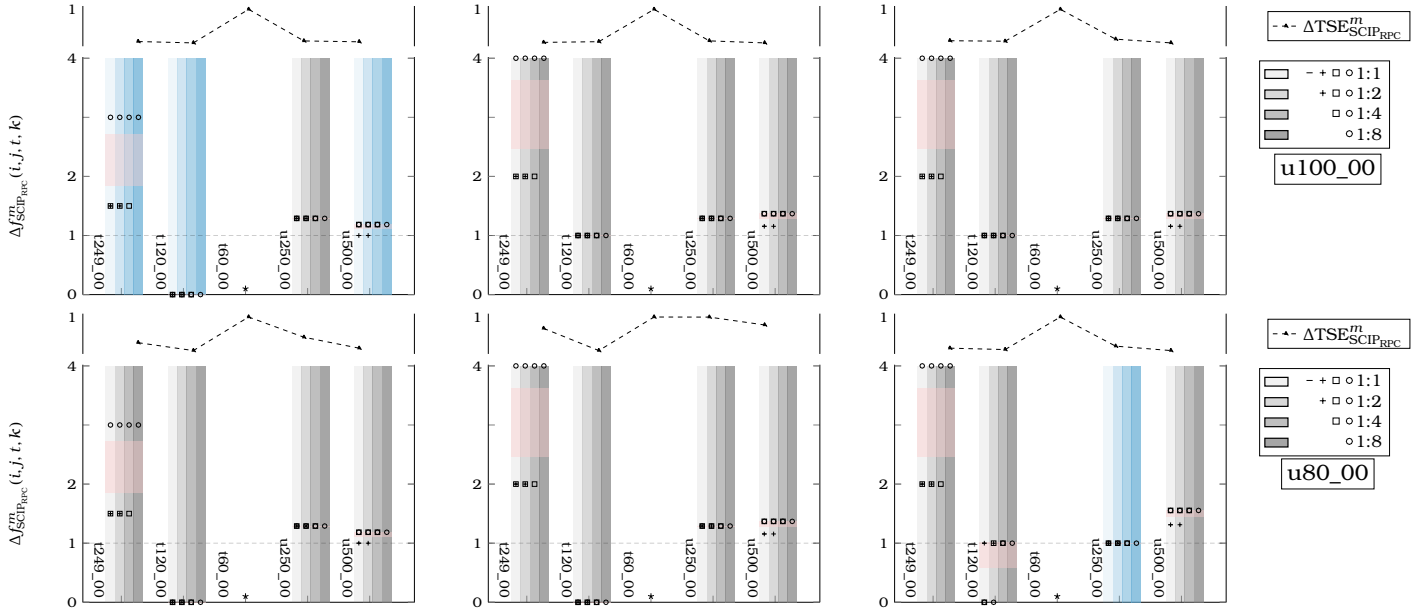


Figure 3.10: The performance of the three BPP classifiers against SCIP_{RPC} calculated by (3.13), across the training instances.

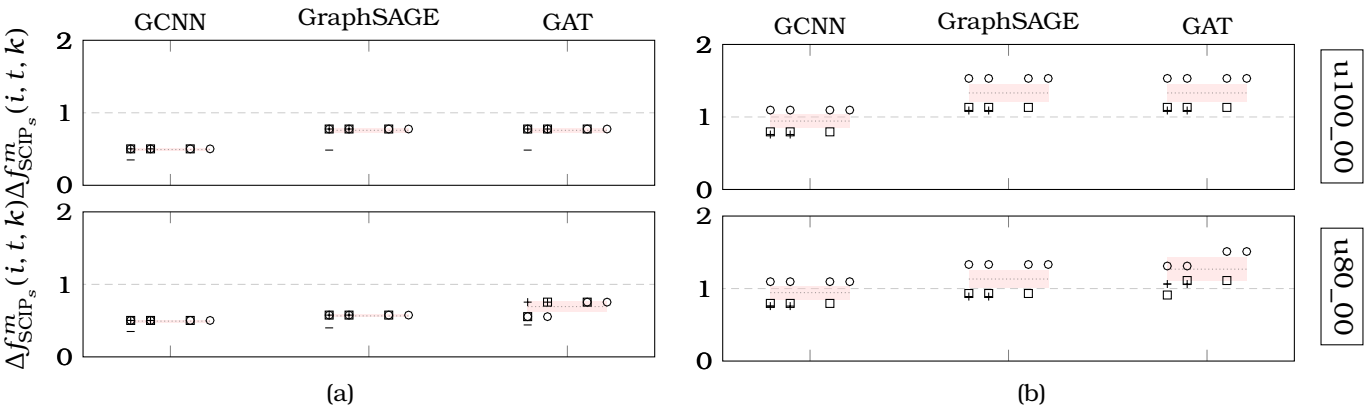


Figure 3.11: The average performance of each classifier across the BPP evaluation instances; (a) compares against SCIP_{SB} and (b) compares against SCIP_{RPC}.

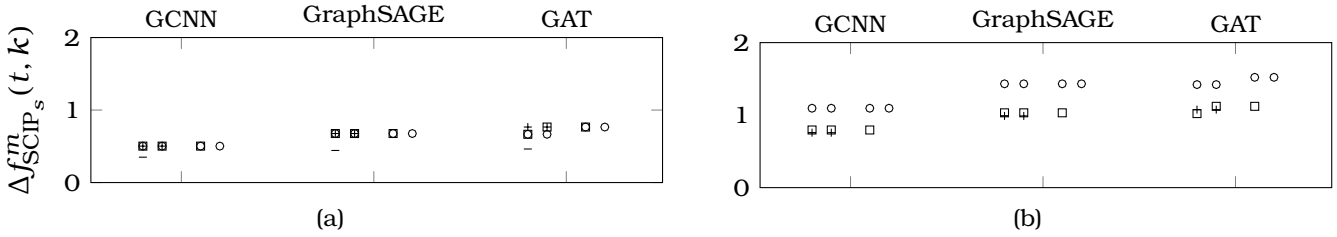


Figure 3.12: The performance of each classifier averaged by both of the BPP training and evaluation instances against (a) SCIP_{SB} and (b) SCIP_{RPC}.

3.7.1 Findings

CVRP

in Figure 3.5, which compares against the performance of SCIP_{SB}, it is evident that the trained classifiers perform better than SCIP_{SB} in the majority of cases, either meeting or surpassing SCIP_{SB}'s performance as indicated by the separating dashed line. Additionally, SCIP_{SB} was unable to find a feasible solution within the first hour for instance P-n76-k5 or within the first, second, and fourth hour for instance B-n57-k7. This trend continues for instance B-n64-k9 across all time windows. Notably, the three classifiers are able to find significantly lower gaps on instance M-n151-k12 even when given the maximum time allowance to SCIP_{SB} (eight hours). Unfortunately, the GED and $\Delta\text{TSE}^{m}_{\text{SCIP}_{\text{SB}}}$ showed fluctuations with no apparent correlation to the classifiers performance, making it challenging to affirm generalization capabilities. When comparing the performance of the individual classifiers, GCNN reaches the lowest gap on four out of eight instances, although the surplus is modest when compared to GraphSAGE. GAT is able to outperform the other two classifiers at the eighth hour, which is emphasized by the average-performance graphs and later explained in this section. It is also worth mentioning that the classifiers highest performance on instances P-n76-k5, B-n78-k10, and M-n151-k12 were trained on A-n33-k6, which may be attributed to a similarity of complexity among them.

The classifiers' performance gains are also reflected in Figure 3.6, comparing against SCIP_{RPC}. Using RPC clearly ameliorate SCIP's performance. Nonetheless, at least one of the classifiers achieves equivalent performance or with slight improvement/decline in at least four hours less. GCNN is still the top-performing classifier on five out of eight evaluation instances. The number of nodes explored is significantly smaller than those of SCIP_{RPC}, this is illustrated in the $\Delta\text{TSE}^{m}_{\text{SCIP}_{\text{RPC}}}$ triangle-dashed line.

The fourth bar ▭ shows especially intriguing results, depicting the performance of the classifiers against a time allowance of eightfold increase for SCIP_{s}. The potential gains when using either branching strategies are minute, and in some cases, there may be a net loss, which may not justify the extra time in time-critical industries.

On average, the classifiers are consistently able to find lower gaps than SCIP_{SB} in significantly less time, as shown in Figure 3.7a, which is calculated using (3.14). Although SCIP_{RPC} improves upon SCIP_{SB}'s performance as depicted in Figure 3.7b, the classifiers still exhibit impressive time savings. These conclusions are further supported by Figure 3.8, which is calculated using (3.15) and demonstrates the classifiers' extrapolation quality and consistency[3]. However, we refrain from using this as a response to research question (C), given the unidentified correlation of the $\Delta\text{TSE}^{m}_{\text{SCIP}_{s}}$ or the GED to the classifier performance and provide further emphasis on the challenge in Section 3.8.

BPP

similar improvements against SCIP_{SB} are apparent in Figure 3.9. There is a noticeable change in patterns, for example, the time limits barely allow for larger optimizations to take place on the selected evaluation instances, either for SCIP_{SB} or any classifier m,

[3]The evaluation instances on which SCIP_{s} failed to find a feasible solution were excluded.

which is observed by the constant values of $\Delta f^{m}_{\text{SCIP}_s}(i,j,t,k)$ across the different values of k. This is also asserted by the non-reported tree-size estimates for three out of five evaluation instances: t249_00, u250_00, and u500_00, which is due to the stale state search tree. We suspect that this is due to obtaining sub-optimal distribution of the bins within a short amount of time and it would require longer time windows to find better assignments. Nonetheless, SCIP_{SB} and each classifiers solved t60_00 to optimality; the former failed to find a feasible solution on t120_00.

For Figure 3.10, SCIP_{RPC} seem to have the advantage on larger time windows for dataset t249_00 with significantly lower gap than that of any of the classifiers, and also successfully obtains a feasible solution on t120_00, but with the lead in performance to at least one of the classifiers that have solved the problem to the optimality.

All of SCIP_{SB} and SCIP_{RPC}, GCNN, GraphSAGE, and GAT found optimal solutions on evaluation instance t60_00 in the first hour.

On average, the classifiers continually secure lower gaps than SCIP_{SB} and comparable performance to SCIP_{RPC}, this is visible in Figures 3.11–3.12. The different training instances hardly make any contribution to the obtained solutions.

3.8 Discussion

In this chapter, we have shown that three geometric deep neural networks: GraphSAGE, GCNN, and GAT are capable of achieving performance that improves upon or matches that of SB on solving CVRP and BPP. The classifiers demonstrate low generalization error on unseen, larger, and seemingly more complex instances. However, we believe that there is a lack of reliable quantitative hardness measurement in the current literature to assess the difficulty of an integer program. This is a challenging task, given their inherent complexity.

While the results were impressive, there are some challenges that may arise in larger instances with hundreds of thousands of customers. These include difficulties in generating decision samples for SB to train a custom model, and limitations in the current approach's ability to generate more efficient strategies than those they were trained on. We suggest two promising directions for these lacunae:

The first option is UG (or ParaSCIP) [55], which is a parallelized implementation of SCIP that allows multiple solvers to explore the search tree simultaneously. UG is a promising direction for enriching our sampling tools and for solving IPs in general. Additionally, we can use transfer-learning from other smaller CVRP (or a different problem altogether) to warm-start the training process, which could help improve the accuracy and efficiency of our classifiers.

The second method is formulating the CVRP as a zero-sum game [56, 57] between two reinforcement-learning agents, with the reward set to maximize the *negative* relative primal-dual gap until a Nash equilibrium is found/approximated [58]. In this approach, one agent "hides" (representing the customers) while the other attempts to "find" it (representing the vehicles). This method has the potential to learn novel methods that improve the performance on solving the CVRP and BPP (and possibly other combinatorial optimization problem), which is our current research focus.

The above contributions open a new research direction which encourages the usages of

geometric deep neural networks to provide real-time good quality solutions to industries within which vehicle routing or crowdsourcing is an essential component, such as last-mile delivery (LMD) and other problems in many other applications.

3.9 Technical Details

The classifiers were trained using distributed training (as a sharded model) on 4 V100 NVIDIA tensor core GPUs for up to 300 epochs. The sampling time took 12-24 hours for the largest instance (M-n151-k12). The computer cluster was SLURM-managed. We designed an automated workflow that preempts the running process and re-queues it for a later continuation without any human involvement. The experiments were monitored using Weights and Biases. The number of experiments executed to examine different possible fine-tuning values is 5000.

Chapter 4

Conclusion

> What can be said at all can be said clearly; and whereof one cannot speak thereof one must be silent.
>
> *Ludwig Wittgenstein*

The main contribution of this research is the development of geometric deep learning classifiers that can provide high-quality solutions to the vehicle routing problem and bin packing problem in time-critical industrial settings. Specifically, the solution to the latter problem should help obviating to "guess" the number of minimum required vehicles in the former, which in turn will reduce the time to finding the solution. Our experiments showed that these classifiers can improve upon or match the performance of SCIP using strong branching or reliable pseudo-cost in a significantly shorter amount of time.

Bibliography

[1] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," *CoRR*, vol. abs/1906.01629, 2019. [Online]. Available: http://arxiv.org/abs/1906.01629

[2] S. P. Boyd, *Convex optimization*. Cambridge, UK: Cambridge University Press, 2004 - 2004.

[3] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning," *Discrete Optimization*, vol. 19, pp. 79–102, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1572528616000062

[4] L. Wosley, *Complexity and Problem Reductions*. John Wiley & Sons, Ltd, 2020, ch. 7, pp. 123–129. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119606475.ch6

[5] R. J. Lipton and K. W. Regan. (2012) Branch and bound–why does it work. [Online]. Available: https://rjlipton.wpcomstaging.com/2012/12/19/branch-and-bound-why-does-it-work/

[6] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *J. ACM*, vol. 7, no. 4, p. 326–329, oct 1960. [Online]. Available: https://doi.org/10.1145/321043.321046

[7] J. J. Hopfield and D. W. Tank, ""neural" computation of decisions in optimization problems," *Biological Cybernetics*, vol. 52, no. 3, pp. 141–152, Jul 1985. [Online]. Available: https://doi.org/10.1007/BF00339943

[8] A. Torki, S. Somhon, and T. Enkawa, "A competitive neural network algorithm for solving vehicle routing problem," *Computers & Industrial Engineering*, vol. 33, no. 3, pp. 473–476, 1997, selected Papers from the Proceedings of 1996 ICC&IC. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S036083529700171X

[9] H. Ghaziri, *Supervision in the Self-Organizing Feature Map: Application to the Vehicle Routing Problem*. Boston, MA: Springer US, 1996, pp. 651–660. [Online]. Available: https://doi.org/10.1007/978-1-4613-1361-8_39

[10] G. Clarke and J. W. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," *Operations Research*, vol. 12, no. 4, pp. 568–581, August 1964. [Online]. Available: https://ideas.repec.org/a/inm/oropre/v12y1964i4p568-581.html

[11] B. E. Gillett and L. R. Miller, "A heuristic algorithm for the vehicle-dispatch problem," *Operations Research*, vol. 22, no. 2, pp. 340–349, 1974. [Online]. Available: https://EconPapers.repec.org/RePEc:inm:oropre:v:22:y:1974:i:2:p:340-349

[12] E. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina, "Learning to branch in mixed

integer programming," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, Feb. 2016. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10080

[13] V. Nair, S. Bartunov, F. Gimeno, I. von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang *et al.*, "Solving mixed integer programs using neural networks," *arXiv preprint arXiv:2012.13349*, 2020.

[14] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, p. 1–122, jan 2011. [Online]. Available: https://doi.org/10.1561/2200000016

[15] G. Zarpellon, J. Jo, A. Lodi, and Y. Bengio, "Parameterizing branch-and-bound search trees to learn branching policies," *CoRR*, vol. abs/2002.05120, 2020. [Online]. Available: https://arxiv.org/abs/2002.05120

[16] P. Gupta, M. Gasse, E. Khalil, P. Mudigonda, A. Lodi, and Y. Bengio, "Hybrid models for learning to branch," *Advances in neural information processing systems*, vol. 33, pp. 18 087–18 097, 2020.

[17] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," 2015. [Online]. Available: https://arxiv.org/abs/1506.03134

[18] R. Bellman, "Dynamic programming treatment of the travelling salesman problem," *J. ACM*, vol. 9, no. 1, p. 61–63, jan 1962. [Online]. Available: https://doi.org/10.1145/321105.321111

[19] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *CoRR*, vol. abs/1611.09940, 2016. [Online]. Available: http://arxiv.org/abs/1611.09940

[20] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takac, "Reinforcement learning for solving the vehicle routing problem," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/9fb4651c05b2ed70fba5afe0b039a550-Paper.pdf

[21] W. Kool, H. van Hoof, and M. Welling, "Attention, Learn to Solve Routing Problems!" *arXiv e-prints*, p. arXiv:1803.08475, Mar. 2018.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[23] 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. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221720306895

[24] p. toth and d. vigo, *the vehicle routing problem*, paolo toth and daniele vigo, Eds. society for industrial and applied mathematics, 2002. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898718515

[25] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. USA: John Wiley & Sons, Inc., 1990.

[26] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959. [Online]. Available: http://www.jstor.org/stable/2627477

[27] J. K. Lenstra and A. H. G. R. Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, 1981. [Online]. Available: https:

//onlinelibrary.wiley.com/doi/abs/10.1002/net.3230110211

[28] S. Arora and B. Barak, *PCP theorem and hardness of approximation: An introduction*. Cambridge University Press, 2009, p. 237–256.

[29] C. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC '88. New York, NY, USA: Association for Computing Machinery, 1988, p. 229–234. [Online]. Available: https://doi.org/10.1145/62212.62233

[30] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian, "New benchmark instances for the capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 257, no. 3, pp. 845–858, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221716306270

[31] Y.-C. Ho and D. Pepyne, "Simple explanation of the no free lunch theorem of optimization," in *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*, vol. 5, 2001, pp. 4409–4414 vol.5.

[32] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," *CoRR*, vol. abs/2104.13478, 2021. [Online]. Available: https://arxiv.org/abs/2104.13478

[33] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960. [Online]. Available: http://www.jstor.org/stable/1910129

[34] D. Applegate, R. Bixby, V. Chvatal, and B. Cook, "Finding cuts in the tsp (a preliminary report)," Tech. Rep., 1995.

[35] T. Achterberg, T. Koch, and A. Martin, "Branching rules revisited," *Operations Research Letters*, vol. 33, no. 1, pp. 42–54, 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167637704000501

[36] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017. [Online]. Available: https://arxiv.org/abs/1710.10903

[37] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *CoRR*, vol. abs/1706.02216, 2017. [Online]. Available: http://arxiv.org/abs/1706.02216

[38] I. Lima, E. Uchoaa, D. Pecinb, A. Pessoaa, M. Poggib, T. Vidal, A. Subramanian, D. Oliveria, and E. Queiroga. (2014) Capacitated vehicle routing problem library. [Online]. Available: http://vrp.galgos.inf.puc-rio.br/

[39] J. E. Beasley. (2004) Operation research library. [Online]. Available: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/

[40] A. Prouvost, J. Dumouchelle, L. Scavuzzo, M. Gasse, D. Chételat, and A. Lodi, "Ecole: A gym-like library for machine learning in combinatorial optimization solvers," in *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020. [Online]. Available: https://openreview.net/forum?id=IVc9hqgibyB

[41] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig, "The SCIP Optimization Suite 8.0," Zuse Institute Berlin, ZIB-Report 21-41, December 2021. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:0297-zib-85309

[42] G. Hendel, D. Anderson, P. Le Bodic, and M. E. Pfetsch, "Estimating the size of

branch-and-bound trees," *INFORMS Journal on Computing*, vol. 34, no. 2, pp. 934–952, 2022. [Online]. Available: https://doi.org/10.1287/ijoc.2021.1103

[43] O. Y. Özaltın, B. Hunsaker, and A. J. Schaefer, "Predicting the solution time of branch-and-bound algorithms for mixed-integer programs," *INFORMS Journal on Computing*, vol. 23, no. 3, pp. 392–403, 2011. [Online]. Available: https://doi.org/10.1287/ijoc.1100.0405

[44] G. Cornuéjols, M. Karamanov, and Y. Li, "Early estimates of the size of branch-and-bound trees," *INFORMS Journal on Computing*, vol. 18, no. 1, pp. 86–96, 2006. [Online]. Available: https://doi.org/10.1287/ijoc.1040.0107

[45] Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, and P. Martineau, "An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems," in *4th International Conference on Pattern Recognition Applications and Methods 2015*, Lisbon, Portugal, Jan. 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01168816

[46] W. L. Hamilton, *The Graph Neural Network Model*. Cham: Springer International Publishing, 2020, pp. 51–70. [Online]. Available: https://doi.org/10.1007/978-3-031-01588-5_5

[47] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016. [Online]. Available: https://arxiv.org/abs/1607.06450

[48] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[49] A. P, N. D, J. M. Belenguer, E. Benavent, A. Corberan, and G. Rinaldi, "Computational results with a branch and cut code for the capacitated vehicle routing problem," Sep 1995.

[50] Y. Pawitan, *In All Likelihood: Statistical Modelling and Inference Using Likelihood*, ser. In All Likelihood: Statistical Modelling and Inference Using Likelihood. OUP Oxford, 2013. [Online]. Available: https://global.oup.com/academic/product/in-all-likelihood-9780199671229

[51] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[52] T. Hastie, R. Tibshirani, and J. Friedman, *Model Assessment and Selection*. New York, NY: Springer New York, 2009, pp. 219–259. [Online]. Available: https://doi.org/10.1007/978-0-387-84858-7_7

[53] V. Feldman, R. Frostig, and M. Hardt, "The advantages of multiple classes for reducing overfitting from test set reuse," *CoRR*, vol. abs/1905.10360, 2019. [Online]. Available: http://arxiv.org/abs/1905.10360

[54] H. Mania, J. Miller, L. Schmidt, M. Hardt, and B. Recht, "Model similarity mitigates test set overuse," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/48237d9f2dea8c74c2a72126cf63d933-Paper.pdf

[55] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, and T. Koch, "Parascip: A parallel extension of scip," in *Competence in High Performance Computing 2010*, C. Bischof, H.-G. Hegering, W. E. Nagel, and G. Wittum, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 135–148.

[56] J. von Neumann and O. Morgenstern, *ZERO-SUM GAMES : THEORY*. Princeton University

Press, 1947, p. 85–165.

[57] J. von Neumann, *1. A Certain Zero-sum Two-person Game Equivalent to the Optimal Assignment Problem*. Princeton: Princeton University Press, 1953, pp. 5–12. [Online]. Available: https://doi.org/10.1515/9781400881970-002

[58] M. Bichler, M. Fichtl, S. Heidekrüger, N. Kohring, and P. Sutterer, "Learning equilibria in symmetric auction games using artificial neural networks," *Nature Machine Intelligence*, vol. 3, no. 8, pp. 687–695, Aug 2021. [Online]. Available: https://doi.org/10.1038/s42256-021-00365-4