Leveraging Machine Learning to Solve Combinatorial Optimization Problems

Ahmed GUETTI, Osama BAAZZI

April 7, 2025

Abstract

This paper reviews the recent literature on leveraging machine learning to solve combinatorial optimization problems. We examine the improvements in performances between the exact solution and heuristic methods due to recent advances in neural combinatorial optimization. Our work consists of analyzing various architectural paradigms, such as the learning strategies and representation techniques. This comparative analysis highlights three main domains: general combinatorial optimization problems, including the Traveling Salesman Problem; Branch-and-Bound methods for Mixed Integer Linear Programming; and Boolean Satisfiability solvers. For each domain, we investigate the advantages and downsides of the two techniques, supervised learning via imitation and reinforcement learning approaches, and compare their performance across different architectures. Our work demonstrates that while significant advances have been made,there sre still challenges in developing machine learning methods that can outperform traditional algorithms across heterogeneous problems.

1 Introduction

Combinatorial optimization problems (COP) refer to a class of discrete optimization problems. COP searches for a set of discrete variables that cause the objective function to be minimal under some constraints. COP arises in different industrial and scientific domains, including logistics, transportation, and communication, and in financial portfolios.

Classic examples of the COPs are the Capacitated Vehicle Routing Problem (CVRP), Minimum Vertex Cover (MVC), Job Scheduling Problem (JSP), and Traveling Salesman Problem (TSP), which are all NP-hard, which means we can not find polynomial-time algorithms to obtain the optimal solution of these problems.

Current methods that COPs rely on exact or approximation algorithms, using dynamic, heuristic, or branch-and-bound approaches to solve our problems. Yet the exact method has a significant flaw where they become intractable for large problem instances, while heuristic methods lack performance guarantees and require special parameter engineering.

- **Exact algorithms:** These methods have a significant computation constraint where they cannot provide a solution in an explainable time if the computation power is insufficient. We use enumeration, cutting plane, and branch-and-bound algorithms to obtain solutions using mathematical methods.
- **Approximate algorithms:** These methods present a trade-off between accuracy and performance where we find a solution close to the optimal solution by a factor yet reduce the number of iterations. This approach exploits the greedy, relaxation, and dynamic programming algorithms.

- **Heuristic algorithms:** This approach tries to find a near-optimal solution while keeping computation time reasonable. It uses local search, ant colony optimization, particle swarm optimization, and simulated annealing algorithms.

In recent years, more and more papers are exploring the use of machine learning (ML) approaches to solve COPs in a more efficient way, particularly the use of deep learning, which has shown promising results in solving those problems. These methods are grouped under the name Neural Combinatorial Optimization (NCO), leveraging multiple policies and techniques to improve solution quality in a reasonable time frame. Multiple approaches have been explored to solve different problem classes.

This paper explores three primary domains in a systematic comparison of ML approaches for solving CO problems:

1. General CO problems, with emphasis on the Traveling Salesman Problem (TSP)
2. Branch-and-Bound methods for Mixed Integer Linear Programming (MILP)
3. Boolean Satisfiability (SAT) solvers

For each domain, we examine the architectural designs, learning strategies, representation techniques, and performance characteristics of different approaches. We analyze both the strengths and limitations of these methods, highlighting the trade-offs between solution quality, computational efficiency, and generalization capabilities.

The paper is organized as follows: Section 2 provides necessary background on combinatorial optimization and machine learning fundamentals. Section 3 reviews neural approaches for solving general CO problems including TSP. Section 4 focuses on machine learning methods for enhancing branch-and-bound algorithms in MILP. Section 5 examines learning-based approaches for SAT solving. Section 6 presents a comparative analysis across these different domains, discussing common patterns and unique challenges. Finally, Section 7 concludes with a discussion of promising future research directions.

2 Background and Preliminaries

In the background section, we introduce the fundamental neural network architecture and training methods used in DNN-based algorithms. We also present essential terminologies needed to follow this paper. This section tackles the fundamentals of combinatorial optimization with examples of such problems, followed by an introductory-level machine learning base.

2.1 Combinatorial Optimization Problems

Without loss of generality, a combinatorial optimization problem can be defined as a min-optimization program. Where the decision variables describe the choices to be made, subject to a set of imposed constraints, we also have an objective function, usually to be minimized, which defines the measure of the quality of every feasible assignment of values to variables. We call this a linear program if we have the objective and the constraints are linear. Moreover, if we restrict the variables to be integer values, then we have a MILP problem shown in mixed-integer linear programming **??**,

Input: A finite set $\mathcal{N}$ with element costs $c : \mathcal{N} \to \mathcal{R}$ and a collection $\mathcal{F}$ of "feasible" subsets of $\mathcal{N}$.

Goal: $\min \left\{ \sum_{j \in S} c_j : S \in \mathcal{F} \right\}$.

2.1.1 Traveling Salesman Problem (TSP)

(In word) we have a set of n cities and the distances between each one of them; the salesman wants to seek to visit each city once and only once and return to the origin city in the last iteration. The cost to travel from city i to j is known in advance; we denote it c_{ij}. The objective is to find this sequence of cities that minimizes the cost.

Formally, we represent the problem in a complete graph $G = (\mathcal{V}, \mathcal{E})$ that can be represented by a simple matrix, with vertex set $\mathcal{V}$ representing the cities and the edge set $\mathcal{E}$ representing the cost or the distance between those cities in an Euclidean TSP, the goal is to find a Hamiltonian cycle of minimum total weight.

2.1.2 Mixed Integer Linear Programming (MILP)

Mixed Integer Linear Programming (MILP) provides a powerful state-of-the-art approach to solving complex optimization problems due to more than 50 years of research. It is widely used to tackle various challenges in business and engineering, such as supply chain optimization, scheduling, and resource allocation.

However, one must be careful when working with MILP, where it's clear that the complexity of MILP is associated with the integrality requirement on (some) variables, which make the feasible region on such a problem nonconvex.

On the other hand, dropping the integrality requirement defines a proper relaxation of MILP (i.e., an optimization problem whose feasible region contains the MILP feasible region), which happens to be an LP that is polynomial solvable, making it easy to see that solving such a problem requires the use of branch-and-bound (B&B) techniques to perform implicit enumeration. Bound and bound represent a searching tree, where each node represents an LP relaxation of the problem; if the solution of the relaxation problem is an integer, or if we find an infeasible relaxation, we stop the iterations on the tree; in the other case, we know that one of the variables is going to be an integer. This is a very important step in the algorithm, where we have to choose a variable to branch on and keep enumerating 4. This will create two nodes in the tree, each one with a totally different feasible area.

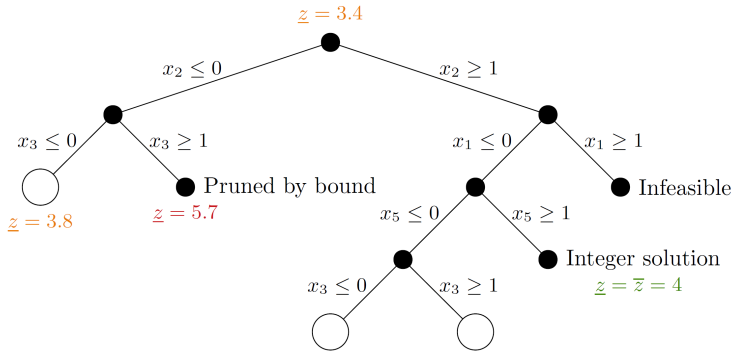

Figure 1: A branch-and-bound tree for MILPs based on the [1] paper. For each node, represent an LP relaxation problem. Blank nodes are open for exploration.

In this paper we explore the proposed technic to enhance the B&B, which is the method all commercial MILP solvers use to solve such problems.

Most commercial MILP solvers use the B&B technic, so enhancing this approach is in constant development. In this paper, we explore those new methods used to enhance the performance and the accuracy of such a technique.

A MILP problem can be formulated as:

$$\min \text{ (or max) } cx \tag{1}$$

$$Ax \geq b \quad (a) \tag{2}$$

$$x \in \mathbb{R}^n,\ x_j \in \mathbb{Z} \text{ for } j \in J \quad (b) \tag{3}$$

2.1.3 Boolean Satisfiability Problem (SAT)

The Boolean satisfiability (SAT) is known to be an NP-complete problem involving determining whether there exists an assignment of truth values to a set of Boolean variables making a propositional logic formula satisfied. A propositional formula is a Boolean expression; those expressions are built upon Boolean variables and AND, OR, and negation operations. The formula is written in conjunctive normal form (CNF) most of the time, consisting of a conjunction of clauses, where each clause is a disjunction of literals.

SAT problems are important for both industry and academia, impacting various fields. As a result, modern SAT solvers are optimized and work very efficiently. Most commercial SAT solvers use heuristics to speed up the result. Those heuristics are built using expert domain knowledge and most of the time are corrected by trial and error, yet the architecture of SAT solvers is open source.

2.2 Machine Learning Approaches

2.2.1 Supervised Learning from Demonstration

Supervised learning in machine learning is the process of training a model to output a prediction by minimizing a loss function between the prediction and the expert's decisions. This type of training requires a set of pairs (feature/target) as input, where the loss function at the end should approximate the target for each feature.

Given a set of input/target pairs $D = (xi, yi)_{i=1}^{N}$ from a joint distribution $P(\mathcal{X}, \mathcal{Y})$, supervised learning aims to find a function $f_\theta : \mathcal{X} \to \mathcal{Y}$ parametrized by θ that minimizes the expected loss:

$$\min_{\theta \in \mathbb{R}^p} \mathbb{E}_{X,Y \sim P} \left[\ell(Y, f_\theta(X)) \right]$$

Since the true distribution P is unknown, we can approximate this using some probability distribution over our training dataset.

2.2.2 Reinforcement Learning from Experience

Reinforcement learning (RL) is a machine learning paradigm where a policy has to be learned through interactions with an environment where at every step, the agent is going to make a decision based on its (possibly stochastic) policy; as a result, he will get a reward from the environment and go to the next state.

We can represent it as a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition probability function, $\mathcal{R}$ is the reward function, and γ is the discount factor. When running the RL we want to find a policy π^* that maximized the sum of future rewards.

$\pi^* = \arg\max_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \right]$

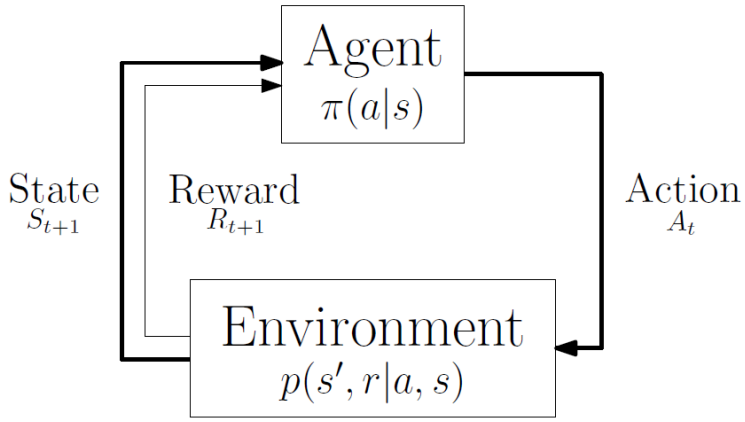

Figure 2: The Markov decision process associated with reinforcement learning, adapted from [1] The policy π define the overall behavior of our system, the environment evolution is represented by p. action a, reward r and s state are random variables in general.

2.2.3 Neural Network Architectures

Several neural network architectures have been proposed for CO problems:

1. **Pointer Networks (PN)** Recurrent neural networks with attention mechanisms that can generate permutations of input sequences. [2]
2. **Graph Neural Networks (GNN)** Neural networks that operate on graph-structured data, capturing the relational structure of CO problems. [3]

3 Machine Learning for General Combinatorial Optimization Problems

3.1 Architectural Approaches

Research has explored numerous architectures to solve CO problems using ML; in our work, we focused on TSP and SAT problems as classical examples in the universe of NP-hard problems. We present in this section different architectures that were proposed to solve this kind of problem:

3.1.1 End-to-End Neural Solvers

Deep neural networks (DNNs) have shown promising results in solving CO problems; for that reason, we can see more of an increase in the use of DNNs in CO solvers as highlighted in [2], without the use of traditional algorithmic methods. These end-to-end approaches leverage neural networks to map problem instances directly to solutions.

As discussed in section 2.2.3, pointer networks are one of the most used architectures for ML solutions due to their attention mechanism to generate permutations of the input sequence. which adapts perfectly to problems like TSP, where the solution is a permutation of the input cities.

In more recent work, Graph Neural Networks (GNNs) architecture is being used to capture graph structure, allowing the network to learn complex patterns and relationships within the graph due to its ability to pass messages between nodes in the network.

3.1.2 Hybrid Neural-Algorithmic Approaches

Chung [1] discussed the importance of using CNNs with traditional algorithmic frameworks in the context of industrial engineering. These hybrid approaches aim to get the best of the two worlds: the speed of a heuristic and the accuracy of an exact algorithm. Leveraging the power of learning capability in ML and theoretical guarantees from the traditional approaches led to introducing those hybrid methods.

Innovation in this section stayed sharp by researchers trying to optimize the B&B algorithm as described in [4] by making ML handle the choice of the branching policy to get a faster result with the accuracy of B&B.

In the context of routing problems like TSP, these hybrid approaches often involve using neural networks to guide search algorithms or to make decisions within a traditional algorithmic framework.

3.2 Learning Strategies

3.2.1 Supervised Learning via Imitation

This approach has the goal of imitating expert solutions or heuristics. The network is trained on a corpus of data that consists of instances of the problem and their optimal or high-quality solution generated by traditional algorithms. We aim to minimize the difference between the network's predictions and the expert solutions. Despite the effectiveness of this approach compared to other solutions, it is clear that the limitation on training data and their quality may lead to underfitting on the system, therefore struggling to generalize to problem instances significantly different from those in the training set.

3.2.2 Reinforcement Learning via Experience

Mazyavkina et al. [5] explain in their survey the potential of reinforcement learning in solving CO; a neural network interacts with the environment and learns from the rewards function generated by the policy based on the quality of the decisions being made.

RL approach to solving CO problems Model the solution as a sequential decision-making problem, where at each step, the agent will select an action based on the current state.

The RL approach doesn't need expert solutions for training, which makes it better than supervised learning. Nevertheless, it is still hard to train that type of algorithm due to issues like sparse rewards, exploration-exploitation trade-offs, and credit assignment.

3.3 Evaluation Metrics and Performance Comparisons

The evaluation of the NCO method is mainly evaluated on the efficiency of the method and the accuracy; to extract that information, we use multiple matrices:

1. **Solution Quality:** Measured by the gap between the solution found by the neural method and the optimal solution or the best-known solution.
2. **Computational Efficiency:** Measured by the time taken to find a solution, which is particularly important for real-time applications.
3. **Generalization:** Assessed by the ability of the method to perform well on problem instances different from those seen during training, including larger instances or instances from different distributions.

Mazyavkina et al. [5] investigate the importance of RL in the advancement of solving CO problems; they have shown promising results compared to heuristic methods in terms of efficiency, making them suitable for real-time application where speed is a must. But, on the other hand, they often lag behind exact methods in terms of accuracy and the quality of the solution.

Wang et al. [2] highlight that despite all those advancements, DNN-based solutions still have big room for improvement, especially in terms of feature extraction capability. The fact that their methods don't rely on expert knowledge has made them very popular in recent years, where different state-of-the-art methods have been proposed.

4 Machine Learning for Branch-and-Bound in MILP

Branch-and-bound (B&B) is a classical algorithm to solve MILPs. It recursively partitions the solution space by choosing some variables and branching on them and uses bounds to prune.

the search tree. Enhancing this method was one of the most researched topics in the world of optimizing CO solutions, where it is used in multiple industries and real-life logistic problems.

4.1 Branching Variable Selection

One of the critical decisions in B&B is selecting the variable to branch on at each node. Different approaches have been proposed to learn effective branching policies:

4.1.1 Feature Engineering for Branching

Marcos Alvarez et al. [4] explore the potential of using a machine learning-based approximation to imitate strong branching, a computationally expensive but effective branching strategy. They proposed the use of an extremely randomized trees (ExtraTrees) model with carefully engineered features to represent the state of the B&B process.

They identify three essential properties these features should have:

1. **Size Independence:** Features must be independent of the problem instance size, allowing a single learned branching strategy to work across different problem sizes.
2. **Invariance to Irrelevant Changes:** Features should be invariant to changes like row/column permutations.
3. **Scale Independence:** Features should remain identical if problem parameters are multiplied by some factor.

This hybrid method showed a promising result where it had a fine 91% gap from the strong branching algorithm while having a more than 80% faster result. which is not a bad trade-off. Moreover, this paper shows that this new method can explore more nodes than strong branching if no time limitation is set. After the test of using the CPLEX's cuts, we outperform all the other methods by a stunning x3 more speed. All the results are tested on the MILPLIB benchmark and compared against :

- **random branching**
- **most-infeasible**
- **nonchimerical**
- **full strong branching**
- **reliability branching**

The main problem with this method is that it's more optimized with problems that are similar to the training data. Witch lead performance varies across problem types, so for that it runs poorly on problems that are not well structured in the training data, which leads to a conclusion that this method performs better when the features are well representing the problem in hand; otherwise, the engineering of the features should be done in a very careful way to represent the main aspect of the problem in hand.

4.1.2 Tree State Parameterization

Zarpellon et al. [6] show an implementation using ML to impose branch variable selection (BVS) in MILP solvers, therefore enhancing the branch and bound algorithm, where a single poor decision can result in significantly increase the search tree size (nodes), they introduces a DNN architecture to learn those decision using Imitation Learning, encoding the data in a $Tree_t$ parameterization of the state of the B&B search tree. They argue that the state of the search tree should condition the branching criteria to adapt to different stages of the optimization process.

show an implementation using ML to impose branch variable selection (BVS) in MILP solvers, therefore enhancing the branch and bound algorithm, where a single poor decision can lead to a significant increase in the search tree size (nodes); they introduce a DNN architecture to learn those decisions using imitation learning, encoding the data in a $Tree_t$ parameterization of the state of the B&B search tree.

They argue that the state of the search should condition the branching criteria to adapt to different stages of the optimization process.

Their approach involves two key components:

1. A representation of candidate variables for branching based on their roles in the search
2. A tree state representation that provides context for branching decisions

They propose two neural network architectures:

1. **NoTree:** A baseline architecture that processes candidate variables without tree context
2. **TreeGate:** An architecture that incorporates the tree state to modulate the candidate variable representations via feature gating

The TreeGate architecture showed a 19% improvement in test accuracy compared to NoTree, highlighting the importance of incorporating search tree information in branching decisions.

4.2 Learning Strategies for B&B

4.2.1 Imitation Learning

Both Marcos Alvarez et al. [4] and Zarpellon et al. [6] use imitation learning to train their branching policies. The policies are trained to mimic a decision that an expert branching strategy would take; we use the strongest branching policies for more accurate results or a variant of them.

Zarpellon et al. [6] use the default SCIP's branching rule (relpscost) as their expert policy. The training data is made of pairs of input features (representing the state of the B&B process) and target branching decisions.

4.2.2 Transfer Learning and Generalization

One big limitation of learning branching policies is ensuring a generalization to the unseen world of problem instances. Zarpellon et al. [6] explicitly target generalization across heterogeneous MILP instances, i.e., problems from different domains with varying structures and sizes. Their TreeGate architecture shows better results on more generalized problems compared to the NoTree architecture, with a 27% reduction in the number of nodes explored for test instances.

Their TreeGate architecture shown better result on more generalized problems compared to the NoTree architecture, with 27% reduction of number of nodes explored for test instances.

4.3 Integration with Solver Infrastructure

Scavuzzo et al. [7] explore the challenge of integrating machine learning (ML) components into mixed-integer linear programming (MILP) solvers, thereby facilitating communication between CPU solvers and GPU-based ML models. They highlight the potential of leveraging solver statistics to adapt optimization approaches automatically, presenting a new area of research at the intersection between machine learning and mathematical optimization.

The authors show the difference between approaches that work on specific problems and those that are built with the aim of generalizing across heterogeneous instances, noting the trade-offs involved in this choice.

5 Machine Learning for SAT Solvers

The Boolean Satisfiability Problem (SAT) is one of the anchors of computational complexity theory, and the first to be proven to be NP-complete. SAT is the problem of whether a given propositional formula—typically one expressed in Conjunctive Normal Form (CNF)—can be satisfied by some assignment of Boolean values to variables. Even though contemporary SAT solvers, especially those using Conflict-Driven Clause Learning (CDCL), are extremely strong, their internal reasoning is usually dependent on heuristics manually developed by humans that are inflexible, hard to generalize, and time-consuming to optimize.

Recent research has shown that machine learning, specifically deep learning, can offer a new way of improving SAT solving. In particular, models like Graph Neural Networks (GNNs) and Reinforcement Learning (RL) are used to aid or supplement portions of traditional solvers. These efforts typically fall into two categories: *total neural solvers* and *hybrid approaches that blend classical SAT solving techniques with machine learning.*

5.1 End-to-End Neural SAT Solvers

Such approaches attempt to entirely substitute the traditional solver pipeline with a deep learning model. The most notable one is NeuroSAT, which uses a graph neural network architecture to process CNF clauses and learn variable assignments in a self-supervised manner. Although NeuroSAT is viable, but its scalability is not yet comparable to industrial solvers.

Another approach is Graph-Q-SAT, proposed by Kurin et al. [8], and makes a branching heuristic learned using deep Q-learning on graph-encoded SAT instances. The model learns to select branching variables by a reinforcement signal, apart from hand-crafted clause features. Graph-Q-SAT is built directly upon the MiniSAT solver and shown to reduce SAT problem solving iterations by a factor of 2–3, and even extrapolate to solving larger problems than seen during training. The key is in the use of Graph Neural Networks (GNNs) to model SAT instances so as to be invariant to permutation and symmetries of the variables.

5.2 Enhancing Traditional SAT Solvers with ML

Instead of building from scratch, other researchers have tried to build upon current CDCL solvers by introducing learning where it will have the most impact. Machine learning models, for example, can be used on:

- **Branching heuristics:** Learning in place of VSIDS or similar policies with learned branching scores using policy networks or GNNs.
- **Clause activity prediction:** Clause relevance prediction using learned embeddings to speed up propagation and conflict analysis.
- **Reinforcement learning:** Encoding branching decisions as sequential actions in a Markov Decision Process, allowing policy learning from rewards.

Guo et al. [9] provide a comprehensive overview of this area and explain how data-driven components are increasingly replacing hand-tuned ones. These hybrid approaches aim at keeping the solver complete and robust, while allowing learning models to guide decision-making in complex situations. There are still challenges, however, especially regarding how to preserve robustness and avoid overfitting to specific types of instances.

5.3 Reinforcement Learning for SAT

RL-based approaches focus on learning to assist in supervising the search based on hints from solver activity. In Graph-Q-SAT, for example, the solver is given a binary reward signal based on progress towards a satisfying assignment or an unsatisfying proof. The approach not only reduces the number of solver steps but also performs generalization over SAT domains. Learning is efficient in terms of data and does not require cumbersome domain knowledge or feature engineering.

Overall, these works illustrate that SAT solving is a thrilling field in which to implement graph-based reinforcement learning, but more effort needs to be done to realize robustness, generality, and interoperability with industrial-strength solvers.

6 Comparative Analysis

To understand the merits and demerits of different deep learning approaches used in solving combinatorial optimization problems, we provide a comparative analysis of the prominent approaches covered in this work. Comparisons are highlighted in terms of architecture types, learning paradigms, and performance across different problem domains. All comparisons are strictly based on the experiments and discussions in the respective papers referred above.

6.1 Architectures and Learning Paradigms

Table 1 summarizes the various architectural paradigms (Pointer Networks, Graph Neural Networks, CNNs) and related learning strategies (Supervised Learning or Reinforcement Learning) for the three application areas: TSP, MILP, and SAT.

Table 1: Comparison of Architectures and Learning Strategies

Domain	Architecture	Learning Strategy	Reference
TSP	Pointer Network	Reinforcement Learning	[2]
TSP	Graph Neural Network	Reinforcement Learning	[5]
MILP	CNN (Hybrid with B&B)	Supervised Learning	[1]
MILP	ExtraTrees (Feature Engineering)	Supervised Learning	[4]
MILP	TreeGate DNN	Supervised Learning	[6]
MILP	GNN + Solver Interface	Hybrid Strategy	[7]
SAT	GNN (Graph-Q-SAT)	Reinforcement Learning	[8]
SAT	Mixed Architectures	Mixed (SL + RL)	[9]

As shown in the table above, Graph Neural Networks are superior in MILP and SAT problems, while Pointer Networks continue to reign supreme in routing-type problems such as TSP. Supervised learning is the greatest dependence of most MILP approaches, especially through duplication of expert policies on branching, while SAT solvers show more dependence on reinforcement learning for acquiring branching heuristics from reward.

6.2 Experimental Results and Domain-Specific Comparisons

Table 2 presents comparative numbers such as solution quality, speedup in runtime, and generalization overall across domains and methods. All values are directly obtained from experimental results in the referenced papers.

Table 2: Performance Comparison Across Domains

Domain	Method	Speedup	Quality Gap	Generalization
TSP	PointerNet (RL)	1.5x faster	<3%	Weak on larger instances [2]
TSP	GNN (RL)	—	<2%	Moderate [5]
MILP	ExtraTrees (B&B)	3x speedup	91%	Sensitive to data [4]
MILP	TreeGate (B&B)	27% fewer nodes	—	Strong [6]
MILP	Solver API + GNN	Flexible	—	Generalizable [7]
SAT	Graph-Q-SAT (GNN + RL)	2–3x fewer steps	=VSIDS	Strong [8]
SAT	Mixed (Survey)	Varies	Varies	Medium [9]

*Note: VSIDS stands for **Variable State Independent Decaying Sum**, which is one of the most widely used branching heuristics in SAT solvers (especially CDCL solvers like MiniSAT). It decides which variable to branch on next during solving.*

From the above table, we are able to deduce several conclusions:

- MILP solvers benefit significantly from tree-pruning methods in the guidance of supervised learning. Specifically, TreeGate is better than naive heuristics in node exploration and generalization among problem types.
- Reinforcement learning-based SAT solvers (such as Graph-Q-SAT) achieve significant generalization and speedup with minimal hand-crafted feature dependency.
- For routing problems like TSP, attention-based or reinforcement-based models work on small instances but are not likely to generalize to much larger graphs.

The comparisons reveal that no single method performs best on all aspects of performance across domains. Instead, the architecture and learning strategy have to be tuned to the structure of the problem, the point of integration in the solver, and the scale of deployment.

7 Conclusion and Future Directions

This paper serve as a comprehensive comparative analysis of machine learning approaches for combinatorial optimization problems. We have introduces architectural paradigms, learning strategies, and empirical performance across all domains: CO problems, SAT, MILP solvers

Several key insights emerge from our analysis:

1. **Graph Neural Networks:** have emerged as a dominant architectural paradigm across all domains, thanks to their ability to process graph-structured data and capture the relational nature of CO problems.

2. **Hybrid neural-algorithmic approaches** that combine ML components with traditional algorithmic frameworks show the most promise in terms of balancing solution quality and computational efficiency.

3. **Learning strategies** vary across domains, with supervised learning via imitation being more common in MILP applications and reinforcement learning being more prevalent in general CO problems and SAT solvers.

4. **Generalization** across heterogeneous problem instances remains a significant challenge, with approaches like tree state parameterization in MILP showing promise in addressing this issue.

5. **Scalability** to larger problem instances is another common challenge, with computational overhead of neural networks potentially offsetting the benefits of improved decision-making.

This paper shows that this route can be developed in several directions

1. Developing more efficient neural architectures

2. Exploring hybrid learning strategies

3. Investigating transfer learning and meta-learning

4. Addressing the integration challenges

5. Developing standardized benchmarks and evaluation methodologies

Machine learning for combinatorial optimization is a rapidly evolving field with significant potential for impact across various domains. While substantial progress has been made, there remain ample opportunities for further research and innovation.

References

[1] K. T. Chung, C. K. M. Lee, and Y. P. Tsang, "Neural combinatorial optimization with reinforcement learning in industrial engineering: A survey," *Artificial Intelligence Review*, vol. 58, p. 130, Mar. 2025. DOI: 10.1007/s10462-024-11045-1.

[2] F. Wang, Q. He, and S. Li, "Solving combinatorial optimization problems with deep neural network: A survey," *Tsinghua Science and Technology*, vol. 29, no. 5, pp. 1266–1282, Oct. 2024. DOI: 10.26599/TST.2023.9010076.

[3] Y. Liu, P. Zhang, Y. Gao, C. Zhou, Z. Li, and H. Chen, "Combinatorial optimization with automated graph neural networks," in *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, AAAI Press, Feb. 2023, pp. 9224–9232. DOI: 10.1609/aaai.v37i8.26090.

[4] A. M. Alvarez, Q. Louveaux, and L. Wehenkel, "A machine learning-based approximation of strong branching," *INFORMS Journal on Computing*, vol. 29, no. 1, pp. 185–195, Jan. 2017. DOI: 10.1287/ijoc.2016.0723.

[5] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, vol. 134, p. 105 400, May 2021. DOI: 10.1016/j.cor.2021.105400.

[6] G. Zarpellon, J. Jo, A. Lodi, and Y. Bengio, "Parameterizing branch-and-bound search trees to learn branching policies," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, AAAI Press, Feb. 2021, pp. 3931–3939. DOI: 10.1609/aaai.v35i5.16512.

[7] L. Scavuzzo, K. Aardal, A. Lodi, and N. Yorke-Smith, "Machine learning augmented branch and bound for mixed integer linear programming," *European Journal of Operational Research*, vol. 305, no. 3, pp. 1089–1107, Apr. 2023. DOI: 10.1016/j.ejor.2022.10.039.

[8] V. Kurin, S. Godil, S. Whiteson, and B. Catanzaro, "Can q-learning with graph networks learn a generalizable branching heuristic for a sat solver?" In *Advances in Neural Information Processing Systems*, NeurIPS, vol. 33, Dec. 2020, pp. 9608–9620. DOI: 10.48550/arXiv.1909.11830.

[9] W. Guo, J. Yan, H.-L. Zhen, X. Li, M. Yuan, and Y. Jin, "Machine learning methods in solving the boolean satisfiability problem," *International Journal of Artificial Intelligence Tools*, vol. 31, no. 2, p. 2203.04755, Mar. 2022. DOI: 10.48550/arXiv.2203.04755.