Contents

Introduction

Combinatorial optimization (CO) occupies a fundamental place in applied mathematics and computer science, seeking optimal solutions within discrete, typically finite, configuration spaces. Formally, a combinatorial optimization problem can be defined as follows: given a finite set S of feasible solutions and an objective function $f : S \to \mathbb{R}$, the aim is to

$$s^* = \arg\min_{s \in S} f(s) \quad \text{or} \quad s^* = \arg\max_{s \in S} f(s),$$

depending on whether f is to be minimized or maximized. CO problems pervade real-world domains, appearing in logistics (e.g., vehicle routing, task scheduling), telecommunications (network design, routing), resource allocation, and task assignment, among many others [1, 2]. These problems are often NP-hard, making exact methods computationally prohibitive for large instances and motivating the search for effective approximation techniques.

Recent advances in deep reinforcement learning (DRL) have opened promising new avenues for tackling such challenging combinatorial problems. DRL integrates reinforcement learning¡[a]a paradigm wherein an agent learns to make a sequence of decisions by maximizing cumulative rewards¡[a]with the expressive power of deep neural networks. In the context of CO, the problem is typically framed as a Markov Decision Process (MDP), where states represent partial solutions, actions correspond to solution construction steps, and the reward reflects solution quality. The agent, parameterized by a deep neural network, learns policies π_θ or value functions V_θ capable of generalizing across problem instances, often surpassing traditional heuristics in both scalability and quality [3, 4].

The intersection of DRL and CO has witnessed a surge of research interest, propelled by DRL's demonstrated success on complex, high-dimensional decision-making tasks. Notable trends include the emergence of neural combinatorial optimization methods, graph-based DRL approaches, and transfer learning strategies to adapt to varying CO landscapes. This rapidly evolving synergy promises not only methodological advances, but also the potential for translating theoretical insights into practical, high-performance solvers for real-world CO problems [3, 4, 5].

This survey is organized as follows. We begin by reviewing foundational concepts in combinatorial optimization and reinforcement learning, then formalize the intersection of these domains. Next, we catalog state-of-the-art DRL methods tailored for CO, describe benchmark problems and evaluation protocols, and analyze empirical results. We conclude by discussing open challenges and prospective research directions.

References

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

[2] Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., and Schrijver, A. Combinatorial Optimization. Wiley, 1998.

[3] Bello, I., Pham, H., Le, Q.V., Norouzi, M., and Bengio, S. Neural Combinatorial Optimization with Reinforcement Learning. arXiv:1611.09940, 2016.

[4] Khalil, E.B., Dai, H., Zhang, Y., Dilkina, B., and Song, L. Learning Combinatorial Optimization Algorithms over Graphs. NeurIPS, 2017.

[5] Joshi, C.K., Cappart, Q., Rousseau, L.-M., Laurent, T., and Bresson, X. Learning TSP Requires Rethinking Generalization. ICLR, 2022.

2. Background

Combinatorial Optimization: Definitions and Classical Approaches

Combinatorial optimization (CO) is the task of finding an optimal object, arrangement, or selection from a finite but typically large set of possibilities, subject to constraints. Formally, given a finite set S and an objective function $f : S \to \mathbb{R}$, the goal is to find $x^* \in S$ such that

$$f(x^*) = \min_{x \in S} f(x)$$

or $\max_{x \in S} f(x)$ for maximization problems [1].

Key Combinatorial Optimization Problems:

- **Traveling Salesman Problem (TSP):**

  Given a set of n cities and a distance matrix $D \in \mathbb{R}^{n \times n}$, find the shortest possible tour that visits each city exactly once and returns to the origin city. This can be formulated as finding a permutation π of the cities that minimizes:

  $$\min_{\pi \in \text{Perm}(n)} \sum_{i=1}^{n} D_{\pi(i), \pi(i+1)}$$

  where $\pi(n+1) = \pi(1)$ [2].

- **Vehicle Routing Problem (VRP):**

  Given a depot, a set of customers with associated demands, and a fleet of vehicles with limited capacity, determine optimal routes for the vehicles such that all customer demands are satisfied with minimum routing cost, and operational constraints (e.g., vehicle capacity, route length) are not violated [3].

- **Graph Coloring Problem:**

  Assign colors to the vertices of a graph $G = (V, E)$ such that no two adjacent vertices share the same color, using the minimum number of colors possible. Formally: $c : V \to \{1, ..., k\}$ where $c(u) \neq c(v)$ for every edge $(u, v) \in E$ [4].

- **Job-Shop Scheduling Problem (JSP):**

  Given a set of jobs, each with an ordered sequence of operations requiring specific machines for given durations, the goal is to schedule all operations on the machines (each can process only one job at a time) to minimize the makespan (the total completion time) [5].

Classical Approaches:

- **Exact Algorithms:**
  - *Branch-and-Bound:* Systematically explores parts of the solution space, using bounds to prune suboptimal regions and guarantee optimality [6].
  - *Dynamic Programming:* Decomposes problems into overlapping subproblems, solving and storing each to avoid redundant calculations, suitable for problems with optimal substructure [7].
- **Heuristics and Metaheuristics:**
  - *Greedy Algorithms:* Construct solutions incrementally, making the locally optimal choice at each step [8].
  - *Local Search:* Begins with an initial solution and iteratively improves it by exploring local neighbors [9].
  - *Simulated Annealing:* A stochastic local search that allows occasional moves to worse solutions to escape local optima, with a temperature parameter controlling the probability [10].
  - *Genetic Algorithms:* Population-based metaheuristics inspired by biological evolution, using operations like selection, crossover, and mutation to evolve better solutions over generations [11].

References

[1] Combinatorial optimization. Wikipedia, accessed 2024.

[2] Travelling salesman problem. Wikipedia, accessed 2024.

[3] P. Toth & D. Vigo (Eds.), *The Vehicle Routing Problem.* SIAM Monographs on Discrete Mathematics and Applications, 2002.

[4] Graph coloring. Wikipedia, accessed 2024.

[5] Job-shop scheduling. Wikipedia, accessed 2024.

[6] Pardalos, P. M., & Resende, M. G. C. (Eds.). "Branch-and-Bound Algorithms." In *Handbook of Applied Optimization*. Oxford University Press, 2002.

[7] Bellman, R. "Dynamic Programming." *Science*, 153(3731), 34-37, 1966.

[8] Feo, T. A., & Resende, M. G. C. "Greedy Randomized Adaptive Search Procedures." *Journal of Global Optimization*, 6, 109-133, 1995.

[9] Russell, S., & Norvig, P. *Artificial Intelligence: A Modern Approach* (4th ed.), Pearson, 2020.

[10] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P., "Optimization by Simulated Annealing." *Science*, 220(4598), 671-680, 1983.

[11] Holland, J. H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

Reinforcement Learning and Deep RL

A Markov Decision Process (MDP) provides the formal foundation for reinforcement learning, modeling sequential decision problems as a tuple (S, A, P, R, γ):

- S is the set of states,
- A is the set of actions,
- $P : S \times A \times S \to [0, 1]$ is the transition probability function $P(s'|s, a)$,
- $R : S \times A \to \mathbb{R}$ is the reward function,
- $\gamma \in [0, 1]$ is the discount factor [12].

Reinforcement Learning Objective and Elements:

The objective in RL is to learn a policy π that maximizes the expected cumulative reward (return) for an agent interacting with an environment modeled as an MDP. A (stochastic) policy $\pi(a|s)$ defines the probability of taking action a in state s [13].

- *Value Function*: The state-value function under policy π, $V^\pi(s)$, is the expected return starting from state s:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s \right]$$

- *Action-Value Function*: $Q^\pi(s, a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, a_0 = a]$
- *Bellman Equation*: The Bellman equation for V^π:

$$V^\pi(s) = \sum_{a \in A} \pi(a|s) \left[R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^\pi(s') \right]$$

Deep Reinforcement Learning (Deep RL):

Deep RL applies deep neural networks to approximate value functions or policies, enabling learning in high-dimensional or unstructured spaces. Key families include:

- **Deep Q-Networks (DQN):** Use neural networks to approximate the action-value (Q) function. DQN selects actions by maximizing the learned Q-values and incorporates techniques like experience replay and target networks for stability [14].
- **Policy Gradients:** Directly parameterize the policy and update its parameters by gradient ascent on expected reward. Useful for high-dimensional or continuous action spaces [15].
- **Actor-Critic:** Combines value-based and policy-based ideas. The "actor" learns the policy and the "critic" estimates value functions, with the critic guiding the actor for more efficient and stable learning [16].

Intuition:

- DQN estimates how good each action is (in terms of future rewards) and chooses accordingly.
- Policy gradients learn a probability distribution over actions directly, updating towards more rewarding actions.
- Actor-critic combines both, enabling scalable and stable learning in complex environments.

References [12] Puterman, M. L., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[13] Sutton, R. S., & Barto, A. G., *Reinforcement Learning: An Introduction* (2nd ed.), MIT Press, 2018.

[14] Mnih, V., et al. "Human-level control through deep reinforcement learning." *Nature* 518, 529¨C533, 2015.

[15] Sutton, R. S., et al., "Policy Gradient Methods for Reinforcement Learning with Function Approximation," NIPS, 2000.

[16] Konda, V. R., & Tsitsiklis, J. N., "Actor-Critic Algorithms," NIPS, 2000.

3. Formulating Combinatorial Optimization as Reinforcement Learning Problems

Combinatorial optimization (CO) problems¡[a]where the goal is to minimize or maximize an objective function $f(x)$ over a discrete set of candidates $x \in \mathcal{X}$¡[a]are classically solved using algorithmic or mathematical programming techniques.

Recent research has shown that framing these problems as sequential decision-making processes allows them to be tackled using deep reinforcement learning (RL) methods [1][2][3][4][5]. This section describes how CO problems are mapped to Markov Decision Processes (MDPs) suitable for RL, discusses challenges in this translation, and provides a detailed example with the Traveling Salesman Problem (TSP).

Markov Decision Process Formulation for CO

An MDP for combinatorial optimization is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, s_0, T)$, where:

- **State space ($\mathcal{S}$):** Each state $s \in \mathcal{S}$ encodes the partial solution constructed so far, as well as additional information about the problem instance (e.g., which elements have been selected or assigned).
- **Action space ($\mathcal{A}$):** Actions correspond to ways of extending or modifying the current partial solution, such as assigning the next variable, selecting an element, or choosing how to complete a structure.
- **Transition function (P):** Given the current state s_t and action a_t, the environment deterministically (or stochastically, if the problem has randomness) transitions to the next state s_{t+1}. For most CO problems, transitions are deterministic and reflect adding an element to the solution.
- **Reward function (R):** The reward provides feedback about solution quality. It is often **sparse** (available only at the end of the episode¡[a]i.e., when a complete solution is built), but can sometimes be designed as **stepwise** (e.g., incremental improvement or penalty at each step). For minimization, negative costs are standard.
- **Start state (s_0):** The initial state corresponds to having constructed no part of the solution (e.g., empty selection, or just the initial node).
- **Termination (T):** Episodes terminate when a complete, valid solution is constructed¡[a]that is, when the stopping criteria of the CO problem are met.

Translating CO Problems into MDPs: Key Considerations

When mapping a CO problem to an RL/MDP framework, several criteria must be considered:

- **State Representation:** How to encode partial solutions such that they contain sufficient information for the agent to make optimal decisions. For example, in routing problems, the state might include the sequence of nodes already visited and the remaining cities.
- **Action Design:** Ensuring that the action space is well-defined (e.g., only feasible choices), and scalable for large problems.

- **Reward Assignment:** Choosing between sparse rewards (assigned at episode end) or potential-shaping with intermediate rewards. The choice impacts agent learning efficiency.
- **Handling Constraints:** Many CO problems have hard constraints. These can be enforced through state/action restrictions or by penalizing infeasible solutions in the reward function [2][3][5].
- **Partial vs. Complete Solutions:** The RL environment typically builds the solution step by step, so the agent's experience at each state is closely linked to the evolving partial solution [2][3].

Worked Example: The Traveling Salesman Problem as an RL/MDP

Problem Statement: Given N cities and a distance matrix $D = [d(i, j)]$, find a tour (a permutation π of the cities) that minimizes the total travel distance:

$$\min_{\pi \in S_N} \left(d(\pi_1, \pi_2) + d(\pi_2, \pi_3) + ... + d(\pi_N, \pi_1) \right)$$

where S_N is the set of all permutations of N elements.

Formulated as an RL/MDP problem:

- **State (s_t):**

  At step t, the state is defined as $s_t = (c_t, V_t)$, where c_t is the current city and V_t is the set of visited cities up to step t.

- **Action (a_t):**

  At each step, the action a_t is the choice of the next unvisited city to visit, i.e., $a_t \in \{1, ..., N\} \setminus V_t$.

- **Transition:**

  Taking action a_t from state s_t moves to $s_{t+1} = (a_t, V_t \cup \{a_t\})$.

- **Reward (r_t):**

  The reward at step t is typically the negative distance traveled:

  $$r_t = -d(c_t, a_t)$$

  Optionally, a terminal reward can be given at episode end for returning to the starting city, i.e., $r_T = -d(c_T, c_0)$.

- **Episode Termination:**

  The episode terminates when all cities have been visited: $|V_t| = N$.

- **Objective:**

  Find a policy π that maximizes the expected cumulative reward (i.e., minimizes the total tour length):

$$\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{N} r_t\right]$$

  where the expectation is with respect to the agent's policy over possible tours [1][2][3].

This mapping enables the use of RL algorithms, such as policy gradient methods, to train agents that incrementally build valid tours while seeking to minimize total travel distance [1][2].

References

[1] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. arXiv:1611.09940, 2016. https://arxiv.org/abs/1611.09940

[2] Nathan Grinsztajn. Reinforcement learning for combinatorial optimization. PhD Thesis, Universit¨¦ Paris Dauphine¨CPSL, 2023. https://theses.hal.science/tel-04353766v1

[3] Y. Liu, X. Li, J. Wang. A Review of Research on Reinforcement Learning for Solving Combinatorial Optimization Problems. CAAI Artificial Intelligence Research, 2023. https://www.sciopen.com/article/10.13568/j.cnki.651094.651316.2023.02.02.0001

[4] Anna Kuzin, Karthik Narasimhan. Reinforcement Learning with Combinatorial Actions. NeurIPS 2020. https://papers.nips.cc/paper/2020/file/06a9d51e04213572ef0720dd27a84792-Paper.pdf

[5] Haoran Chen, Rui Song, Weinan Zhang. Reinforcement learning for combinatorial optimization: A survey. Computers & Operations Research, 2021. https://www.sciencedirect.com/science/article/abs/pii/S0305054821001660

4. Survey of Recent Deep Reinforcement Learning Methods for Combinatorial Optimization

Overview Table/List

The following table summarizes some of the most influential deep reinforcement learning (DRL) papers on combinatorial optimization (CO) from 2017 to 2024, including the CO problems addressed and key DRL techniques used.

Citation	Problem(s)	DRL Technique
[1] Bello et al., "Neural Combinatorial Optimization with Reinforcement Learning," 2017	TSP, Knapsack	Pointer Network, Policy Gradient
[2] Kool et al., "Attention, Learn to Solve Routing Problems!", 2019	TSP, VRP, Orienteering	Attention Model (Transformer), Policy Gradient
[3] Joshi et al., "Learning TSP Requires Rethinking Generalization," 2021	TSP	GNN, Supervised and RL
[4] Cappart et al., "Combinatorial Optimization and Reasoning with Graph Neural Networks," 2021	Max-Cut, MWIS, MVC, SAT	GNN, RL
[5] Nguyen et al., "RL4CO: Benchmarking Reinforcement Learning for Combinatorial Optimization," 2024	Various CO (TSP, VRP, Knapsack, etc.)	RL Benchmark Study

Model Architectures and Training Algorithms

Pointer Networks

Pointer Networks [1] are sequence-to-sequence models that allow the output to be a permutation of the input, making them suitable for CO problems with ordering (e.g., TSP). The network outputs a sequence π by attending over inputs at each decoding step.

- **Input:** Sequence of node embeddings $X = (x_1, ..., x_n)$ (e.g., city coordinates).
- **Output:** Permutation $\pi = (\pi_1, ..., \pi_n)$ of node indices.
- **RL Objective:** Maximize expected reward (e.g., negative tour length):

$$J(\theta) = \mathbb{E}_{\pi \sim p_\theta(\cdot|X)}[R(\pi|X)]$$

- **Policy Gradient Update:**

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[R(\pi|X) \nabla_\theta \log p_\theta(\pi|X)]$$

Attention Models (Transformers)

Kool et al. [2] proposed attention-based models for routing. These use self-attention to represent input graphs, improving generalization and scalability.

- **Input:** Same as pointer network.
- **Output:** Sequence decision via autoregressive decoding with masked softmax over nodes.
- **Objective and updates:** As above, typically with REINFORCE or rollout baseline variance reduction.

Graph Neural Networks (GNNs)

GNNs [3,4] represent CO problems on graphs (nodes, edges), enabling message passing and flexible architectures for non-sequential tasks.

- **Input:** Graph $G = (V, E)$ with features.
- **Output:** Problem-dependent (e.g., label for each node/edge, or selection mask).
- **RL Objective:**

$$J(\theta) = \mathbb{E}_{a \sim \pi_\theta(\cdot|G)}[R(a|G)]$$

where a is an action (e.g., subset of nodes/edges).

Policy Gradient and Actor-Critic Methods

Policy gradient [1,2] and actor-critic [4] are used to train models using sampled solution rewards.

- **Policy Gradient:**

$$\nabla_\theta J(\theta) = \mathbb{E}[R(s) \nabla_\theta \log p_\theta(a|s)]$$

- **Actor-Critic Update:** The critic estimates value $V_\phi(s)$:

$$L_{critic} = \|R(s) - V_\phi(s)\|^2$$

The actor is updated with reduced-variance advantage:

$$A(s, a) = R(s) - V_\phi(s)$$

Problem-Specific Applications and Case Studies

TSP and VRP

- **Problem Statement:**
  - **TSP**: Given n cities with coordinates X, find a permutation π minimizing total length:

$$\min_{\pi \in S_n} \sum_{i=1}^{n} d(x_{\pi_i}, x_{\pi_{i+1}})$$

    where d is the Euclidean distance.
  - **VRP**: Extension where a vehicle visits nodes subject to capacity constraints.
- **DRL Solution:**
  - MDP setup: State = partial tour/route, Action = select next node, Reward = negative additional cost.
  - Model/Algorithm: Pointer network [1] or attention model [2], trained with policy gradient. Rollout baseline used for variance reduction.
  - Empirical findings: These approaches learn from scratch, outperform some heuristics, and generalize to larger instances better than supervised models [2].

Graph Problems (Coloring, Max-Cut, MVC, etc.)

- **Problem Statement:**
  - **Max-Cut**: For graph $G = (V, E)$, partition V to maximize cut weight:

$$\max_{S \subseteq V} \sum_{(u,v) \in E} w_{uv} \cdot \mathbf{1}[u \in S, v \notin S]$$

  - **MVC/MWIS**: Find minimum vertex cover or maximum weight independent set.
- **DRL Solution:**
  - MDP setup: State = current partial solution, Action = node/edge selection, Reward = change in objective value.
  - Model: GNN-based policy [4], trained with actor-critic.
  - Empirical findings: GNN-RL solutions outperform classic heuristics on diverse graphs, can be retrained for new objectives [4].

Scheduling/Other Problems

- **Problem Statement:**
  - E.g., Job-shop scheduling: Assign tasks T to machines M to minimize makespan.
- **DRL Solution:**
  - MDP: State = current schedule, Action = assign next task, Reward = negative increase in makespan.
  - Model: GNN or attention networks, trained via policy gradient or actor-critic.
  - Empirical findings: DRL can adapt to dynamic changes and constraints, with competitive performance but higher variance [5].

References

[1] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. NeurIPS, 2017.

[2] Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems! ICLR, 2019.

[3] S. V. N. Vishwanathan, Y. Bengio, et al. Learning TSP Requires Rethinking Generalization. ICLR, 2021.

[4] Quentin Cappart, Laurent Prud¡¯Homme, Andr¨¦ Cire, et al. Combinatorial Optimization and Reasoning with Graph Neural Networks. Constraints, 2021.

[5] X. Nguyen, A. Kurin, et al. RL4CO: Benchmarking Reinforcement Learning for Combinatorial Optimization. 2024.

5. Comparative Analysis and Discussion

Deep reinforcement learning (DRL) has recently emerged as a promising approach for combinatorial optimization (CO) problems, challenging the dominance of classical algorithms, including exact, heuristic, and metaheuristic methods. This section presents an in-depth comparative analysis, referencing previous survey findings and state-of-the-art literature, with attention to empirical performance, mathematical/algorithmic trade-offs, and open challenges.

Empirical Comparison: Solution Quality, Computation Time, Generalization, Robustness, and Scalability

Common Benchmarks

- **Traveling Salesman Problem (TSP) & Vehicle Routing Problem (VRP):** Classical exact methods (e.g., Concorde for TSP) can solve small-to-medium instances to optimality, but computational cost grows exponentially with problem size. Metaheuristics (e.g., LKH, Simulated Annealing) provide near-optimal solutions efficiently for larger instances. DRL methods such as the Pointer Network [1], attention models [2], and RL4CO benchmark [5] have shown that:
  - On small-to-moderate graphs (up to 100 nodes), DRL-based solvers match or even outperform some heuristics in solution quality, but typically do not surpass highly optimized metaheuristics like LKH for TSP [2].
  - On larger graphs, DRL inference is fast (amortized after training), but model training is computationally intensive [5].
- **Graph Problems (MaxCut, Minimum Vertex Cover, MWIS, etc.):** Classical algorithms include greedy, spectral, local search, and semidefinite programming (SDP) relaxations. DRL approaches, especially GNN-based [4], statistically outperform simple heuristics on diverse random graphs, but their solutions often lag behind problem-specific metaheuristics and exact solvers for large or specially structured instances [4].

Strengths Revealed by Benchmarks

- **Generalization:** Attention-based and GNN models display strong generalization to unseen problem sizes and distributions, in contrast to classical heuristics that may require manual adaptation [2,3]. However, generalization often degrades significantly for input distributions far from the training set [3,5].
- **Robustness:** DRL models can adapt to noisy or dynamically changing environments via retraining or fine-tuning, while many classical heuristics lack this flexibility.
- **Scalability and Inference Time:** Once trained, DRL policies provide very fast solution inference (constant time per instance), outperforming exact solvers and some heuristics in run-time for large-scale, real-time applications [1,2]. However, training time and data requirements are substantial [5].

Limitations Relative to Classical Methods

- **Solution Optimality:** DRL techniques rarely achieve the best-known or optimal solutions on benchmark instances when compared to dedicated metaheuristics or exact methods (e.g., LKH for TSP, SDP for MaxCut) [2,4].
- **Computation Cost:** Training DRL models requires significant computational resources and careful hyperparameter tuning; by contrast, classical methods work out-of-the-box with much lower startup cost.
- **Reproducibility:** Variance in results due to random initialization and sensitivity to training hyperparameters can hinder reproducibility, while classical algorithms¡¯ outputs are usually deterministic given the same instance and seed.

Mathematical and Algorithmic Strengths and Weaknesses

DRL Approaches

- **Strengths:**
  - End-to-end parameterization allows direct optimization of combinatorial objectives via gradient-based RL, overcoming the need for handcrafted rules.
  - Neural architectures (e.g., attention, GNNs) can flexibly model a range of CO problems, enabling transfer and meta-learning.
  - Capable of integrating problem-adaptive embeddings for learning complex policies.
- **Weaknesses:**
  - Objective landscapes are non-convex, with possible local optima and unstable training.
  - Require large amounts of training data; learning is sample-inefficient when compared to most classical techniques.
  - Interpretability is limited relative to structured human-designed algorithms.

Classical Approaches (Exact, Heuristic, Metaheuristic)

- **Strengths:**
  - Mathematical guarantees (optimality, bounding) for exact algorithms (e.g., branch-and-bound, dynamic programming).

  - Heuristics/metaheuristics are highly engineered, robust, and can incorporate domain-specific knowledge through clever rules or local search strategies.
  - Reproducible, transparent, and often require little training or parameter tuning.
- **Weaknesses:**
  - Scalability: Exact algorithms¡¯ complexity grows exponentially; heuristics, while faster, may not generalize without adaptation.
  - Lack adaptability to nonstationary or highly dynamic environments unless extended by learning components.

Open Challenges in DRL for CO

- **Sample Efficiency:** DRL approaches are typically data-hungry, needing millions of sample trajectories for competitive performance [5]. Bridging this gap with techniques like curriculum learning and self-play is ongoing research.
- **Interpretability:** Neural policies are often opaque, making it difficult to extract and formalize solution strategies or guarantee feasibility.
- **Reproducibility:** Training instability and reliance on random seeds/hyperparameter choices reduce reproducibility¡ªconsistent benchmarking initiatives (e.g., RL4CO [5]) attempt to address this issue.
- **Constraint Handling:** Implicit constraint satisfaction remains a challenge; encoding hard constraints directly into neural architectures or via constraint programming hybrids is an active area.
- **Generalization to Out-of-Distribution Instances:** Generalization remains limited when test instances differ significantly from those seen during training; meta-learning and robust optimization approaches are being investigated [3,5].

References

[1] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. NeurIPS, 2017.

[2] Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems! ICLR, 2019.

[3] S. V. N. Vishwanathan, Y. Bengio, et al. Learning TSP Requires Rethinking Generalization. ICLR, 2021.

[4] Quentin Cappart, Laurent Prud¡¯Homme, Andr¨¦ Cire, et al. Combinatorial Optimization and Reasoning with Graph Neural Networks. Constraints, 2021.

[5] X. Nguyen, A. Kurin, et al. RL4CO: Benchmarking Reinforcement Learning for Combinatorial Optimization. 2024.

Conclusion and Future Directions

Deep reinforcement learning (DRL) for combinatorial optimization (CO) has rapidly advanced, driven by innovations in neural architectures and reinforcement learning algorithms. This survey synthesized the foundations of CO and RL, detailed the mapping of CO to RL frameworks, reviewed leading DRL models and techniques, and presented an empirical and algorithmic comparative analysis. Below, we summarize the main findings and outline outstanding challenges and promising directions for future research in this growing field.

Summary of Main Findings

- **Expressivity and Flexibility:** DRL models, especially pointer networks, attention-based architectures, and graph neural networks (GNNs), can flexibly represent and solve a wide range of CO problems without hand-crafted rules, handling sequences (e.g., TSP, VRP) and graph-based instances (e.g., Max-Cut, vertex cover) alike [1,2,4].
- **End-to-End Learning:** Modern DRL approaches learn policies directly from data via gradient-based optimization. Policy gradient and actor-critic algorithms have proven effective in optimizing combinatorial objectives in discrete action spaces [1,2,4,5].
- **Benchmark Performance:** Empirical studies show that, on small to medium-sized problems (typically up to a few hundred nodes), DRL approaches achieve solution quality competitive with or superior to classical heuristics and sometimes near metaheuristics [1,2,5]. Inference from trained policies is much faster than from exact solvers or metaheuristics, supporting applications in real-time and large-scale settings [1,2].
- **Generalization and Adaptability:** Neural DRL models display strong generalization to larger problem instances and can adapt to changes in problem distributions or constraints through retraining or fine-tuning [2,4].

State-of-the-Art and Current Limitations

Despite remarkable progress, DRL for CO remains an area with challenging open problems:

- **Optimality Gap:** DRL methods rarely achieve the tightest-known or provably optimal solutions, especially on large or structured instances,

lagging behind specialized metaheuristics and exact algorithms (e.g., LKH for TSP, SDP relaxations for Max-Cut) [2,4].

- **Sample Inefficiency:** State-of-the-art DRL approaches require large numbers of training samples to learn effective policies, which leads to substantial computation time and energy consumption [5].
- **Reproducibility and Stability:** Results are sensitive to initialization and hyperparameters, with notable variance across training runs. Benchmarking frameworks (e.g., RL4CO) are helping to stabilize comparisons [5].
- **Interpretability:** The decision processes learned by deep policies are generally opaque, complicating diagnosis, debugging, and the extraction of theoretical insights.
- **Constraint Handling:** Many CO problems feature complex constraints that DRL models often learn to satisfy only implicitly or imperfectly; integrating explicit constraint reasoning remains difficult [5].

Promising Research Directions

The intersection of DRL and CO is poised for further advances in several critical areas:

- **Improved Sample Efficiency:** Techniques such as curriculum learning, experience replay, model-based RL, and leveraging synthetic or transfer datasets could significantly reduce training costs [5].
- **Generalization and Robustness:** Research into meta-learning, domain adaptation, and robust optimization aims to enable DRL models to generalize reliably to out-of-distribution and dynamic environments [3,5].
- **Interpretability and Theory-Guided RL:** Making DRL policies more transparent¡[a]possibly through hybrid designs with classical algorithmic components or explicit solution construction rules¡[a]can foster trust, safety, and theoretical understanding [4].
- **Hybrid and Composable Methods:** Combining strengths of classic heuristics (e.g., local search, constraint programming) with DRL through hybrid or modular approaches can improve both performance and reliability, especially for large-scale and highly constrained problems.
- **Scalable and Efficient Architectures:** Further innovations in neural architectures (e.g., scalable GNNs, efficient attention mechanisms) and scalable training paradigms (e.g., distributed RL) may address the bottlenecks of current models [2,4].
- **Application to New Domains:** As DRL models grow more flexible, new and complex CO problems (e.g., energy systems, logistics, bioinformatics)

may become accessible to learning-based discovery and optimization, potentially expanding the practical impact of the field.

Open Questions

Major open questions motivating ongoing research include:

- How can we close the gap in solution optimality between DRL methods and state-of-the-art metaheuristics on difficult benchmarks?
- Can sample efficiency and training time be improved to reach practical deployment in industrial settings?
- What are the limits of out-of-distribution generalization for learned policies, and how can they be pushed further?
- How can we build DRL models with stronger theoretical guarantees, interpretability, and reliability, especially under real-world constraints?

In summary, deep reinforcement learning for combinatorial optimization offers remarkable expressiveness and adaptability, but significant theoretical, practical, and engineering challenges remain. Future breakthroughs will likely arise from interdisciplinary work uniting advances in learning algorithms, combinatorial optimization, theory, and robust engineering.

References

[1] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. NeurIPS, 2017.

[2] Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems! ICLR, 2019.

[3] S. V. N. Vishwanathan, Y. Bengio, et al. Learning TSP Requires Rethinking Generalization. ICLR, 2021.

[4] Quentin Cappart, Laurent Prud¡¯Homme, Andr¨¦ Cire, et al. Combinatorial Optimization and Reasoning with Graph Neural Networks. Constraints, 2021.

[5] X. Nguyen, A. Kurin, et al. RL4CO: Benchmarking Reinforcement Learning for Combinatorial Optimization. 2024.