Papers on ML4CO

Gemini Light
wtfly2018@163.com

`Lasted Update: 2021/08/20`
`Current Version: v0.2 (P)`

Preface

This is a brief survey on machine learning for combinatorial optimization, a fascinating and significant topic. We mainly collect relevant papers from top conferences, classify them into several categories by method types, and indicate additional information.

It's inevitable to have omissions or errors during the discussion for the limitation of the author's knowledge. Welcome to feedback them you found.

To-Do List

- Continue to collect high-quality papers on ml4co.
- Add formal definitions and practical applications of every problem.
- Specify problem modelling in representative papers.
- Provide more detail for each model.
- Supply more analysis of development.

Combinatorial Optimization Problems

- **TSP**: Travelling Salesman Problem
- **VRP**: Vehicular Routing Problem
- **MAXCUT**: Maximum Cut
- **MVC**: Minimum Vertex Cover
- **MIS**: Maximal Independent Set
- **MC**: Maximal Clique
- **MCP**: Maximum Coverage Problem
- **SAT**: Satisfiability
- **GC**: Graph Coloring
- **JSSP**: Job Shop Scheduling Problem

- **3D-BPP**: 3D Bin-Packing Problem
- **MIP/MILP**: Mixed Integer (Linear) Programming

Categories of Machine Learning

- **SL**: Supervised Learning
- **UL**: Unsupervised Learning
- **RL**: Reinforcement Learning *(Including Imitation Learning, IL)*

Components of Neural Network

- **RNN**: Recurrent Neural Network *(Including LSTM, GRU)*
- **CNN**: Convolutional Neural Network
- **GNN**: Graph Neural Network *(Including Graph Embedding, GE)*
- **Attention** *(Including Self-Attention)*
- **Transformer**

Content

Survey

1. **On Learning and Branching: a Survey**
   - `Publication` : TOP 2017
   - `Keyword` : Branch
   - `Link` : paper
2. **Boosting Combinatorial Problem Modeling with Machine Learning**
   - `Publication` : IJCAI 2018

- `Keyword` : ML
- `Link` : arXiv

3. **A Review of Combinatorial Optimization with Graph Neural Networks**
   - `Publication` : BigDIA 2019
   - `Keyword` : GNN
   - `Link` : paper
4. **Learning Graph Matching and Related Combinatorial Optimization Problems**
   - `Publication` : IJCAI 2020
   - `Keyword` : GNN, Graph Matching
   - `Link` : paper
5. **Learning Combinatorial Optimization on Graphs: A Survey with Applications to Networking**
   - `Publication` : IEEE ACCESS 2020
   - `Keyword` : GNN, Computer Network
   - `Link` : paper
6. **A Survey on Reinforcement Learning for Combinatorial Optimization**
   - `Publication` : arXiv 2020
   - `Keyword` : RL
   - `Link` : arXiv
7. **A Survey on Reinforcement Learning for Combinatorial Optimization**
   - `Publication` : arXiv 2020
   - `Keyword` : RL
   - `Link` : arXiv
8. **Reinforcement Learning for Combinatorial Optimization: A Survey**
   - `Publication` : arXiv 2020
   - `Keyword` : RL
   - `Link` : arXiv
9. **Graph Learning for Combinatorial Optimization: A Survey of State-of-the-Art**
   - `Publication` : Data Science and Engineering 2021
   - `Keyword` : GNN
   - `Link` : arXiv
10. **Combinatorial optimization and reasoning with graph neural networks**
    - `Publication` : arXiv 2021
    - `Keyword` : GNN
    - `Link` : arXiv

Analysis

1. **Learning to Branch**
   - `Publication` : ICML 2018

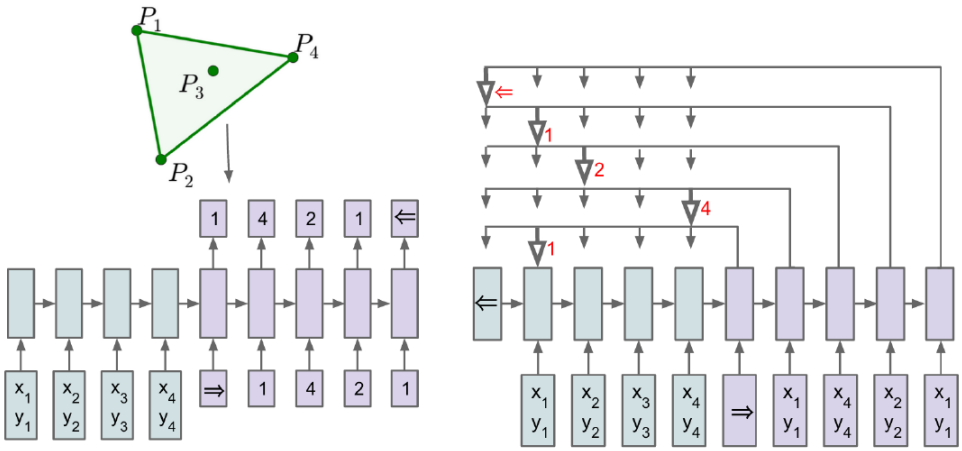
- `Keyword` : Branch
- `Link` : arXiv

2. **Approximation Ratios of Graph Neural Networks for Combinatorial Problems**
   - `Publication` : NeurIPS 2019
   - `Keyword` : GNN
   - `Link` : arXiv
3. **On Learning Paradigms for the Travelling Salesman Problem**
   - `Publication` : NeurIPS 2019 (Workshop)
   - `Keyword` : RL vs SL
   - `Link` : arXiv
4. **On the Difficulty of Generalizing Reinforcement Learning Framework for Combinatorial Optimization**
   - `Publication` : ICML 2021 (Workshop)
   - `Keyword` : GNN
   - `Link` : arXiv

End-to-end

RNN/Attention-based

1. **Pointer Networks**
   - `Publication` : NeurIPS 2015
   - `CO-problem` : Finding planar convex hulls, computing Delaunay triangulations, TSP
   - `ML-type` : SL
   - `Component` : LSTM, Seq2Seq, Attention
   - `Innovation` : Ptr-Net not only improve over sequence-to-sequence with input attention, but also allow us to generalize to variable size output dictionaries.
   - `Link` : arXiv



(a) Sequence-to-Sequence (b) Ptr-Net

Figure 1: **(a)** *Sequence-to-Sequence* - An RNN (blue) processes the input sequence to create a code vector that is used to generate the output sequence (purple) using the probability chain rule and another RNN. The output dimensionality is fixed by the dimensionality of the problem and it is the same during training and inference [1]. **(b)** *Ptr-Net* - An encoding RNN converts the input sequence to a code (blue) that is fed to the generating network (purple). At each step, the generating network produces a vector that modulates a content-based attention mechanism over inputs ([5, 2]). The output of the attention mechanism is a softmax distribution with dictionary size equal to the length of the input.

2. **Neural Combinatorial Optimization with Reinforcement Learning**

- `Publication` : ICLR 2017
- `CO-problem` : TSP
- `ML-type` : RL (Actor-critic)
- `Component` : LSTM, Seq2Seq, Attention
- `Innovation` : This paper presents a framework to tackle combinatorial optimization problems using neural networks and reinforcement learning.
- `Link` : arXiv

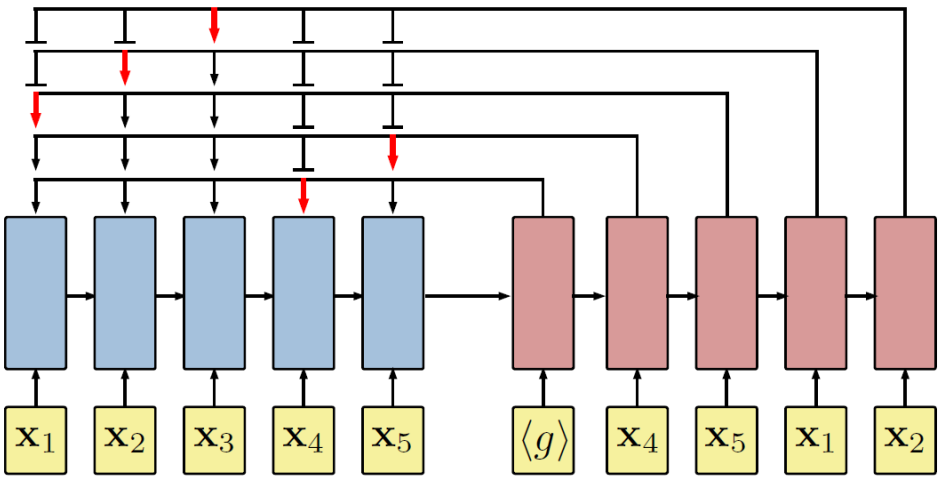


Figure 1: A pointer network architecture introduced by (Vinyals et al., 2015b).

3. **Multi-Decoder Attention Model with Embedding Glimpse for Solving Vehicle Routing Problems**

- `Publication` : AAAI 2021
- `CO-problem` : VRP, TSP
- `ML-type` : RL (REINFORCE)
- `Component` : Self-Attention

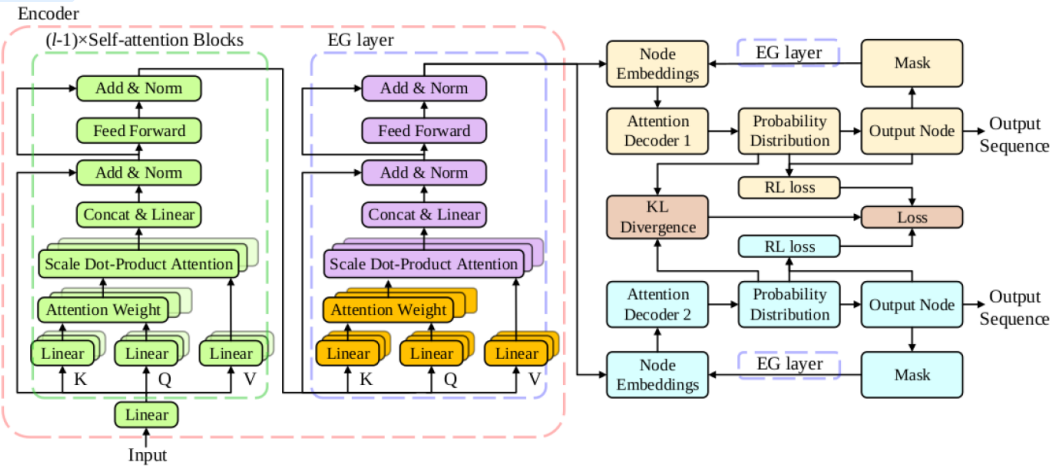
- `Innovation` : This paper proposes a Multi-Decoder Attention Model (MDAM) to train multiple diverse policies, which effectively increases the chance of finding good solutions compared with existing methods that train only one policy. A customized beam search strategy is designed to fully exploit the diversity of MDAM.
- `Link` : arXiv
- 

Figure 1: The Multi-Decoder Attention Model (we use two decoders for illustration)

4. **Learning Improvement Heuristics for Solving Routing Problems**
   - `Publication` : TNNLS 2021
   - `CO-problem` : TSP, VRP
   - `ML-type` : RL (n-step Actor-Critic)
   - `Component` : Self-Attention
   - `Innovation` : This paper proposes a self-attention based deep architecture as the policy network to guide the selection of next solution.
   - `Link` : arXiv

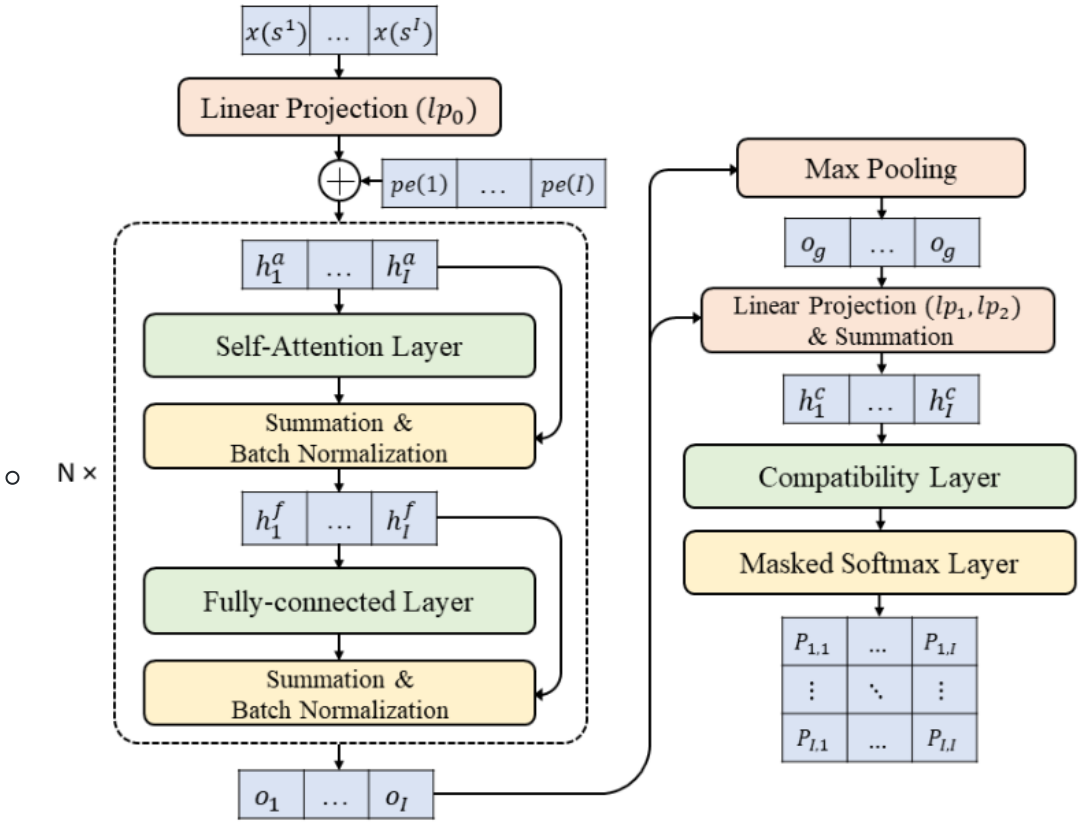


Fig. 2: The architecture of policy network (left: node embedding; right: node pair selection)

5. **The Transformer Network for the Traveling Salesman Problem**
   - `Publication` : arXiv 2021
   - `CO-problem` : TSP
   - `ML-type` : RL
   - `Component` : Transformer
   - `Innovation` : This paper proposes to adapt the recent successful Transformer architecture originally developed for natural language processing to the combinatorial TSP.
   - `Link` : arXiv
   -

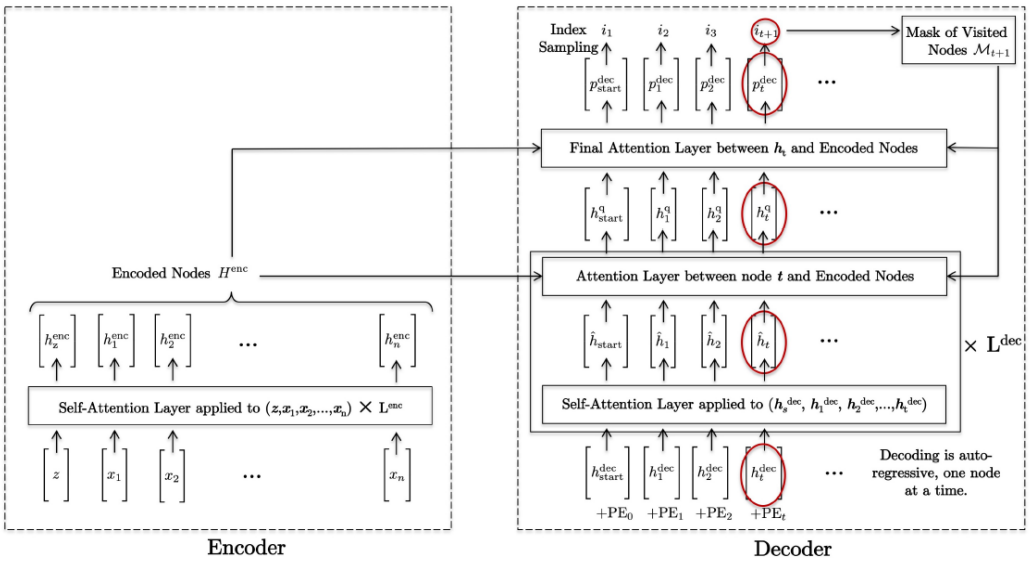


Figure 1: Proposed TSP Transformer architecture.

6. **Matrix Encoding Networks for Neural Combinatorial Optimization**

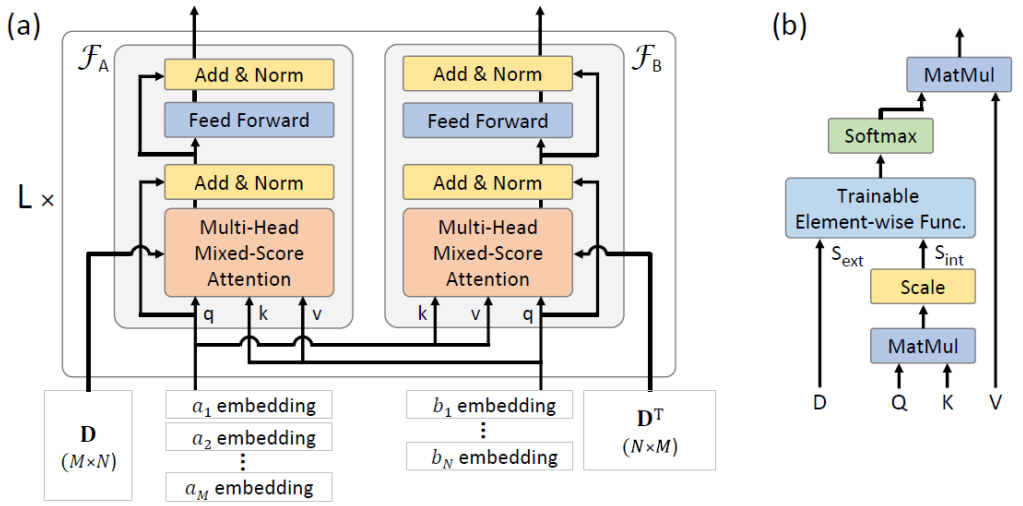
- `Publication` : arXiv 2021
- `CO-problem` : TSP
- `ML-type` : RL (REINFORCE)
- `Component` : Attention
- `Innovation` : MatNet is capable of encoding matrix-style relationship data found in many CO problems
- `Link` : arXiv
- 

Figure 2: (a) An overview of the MatNet architecture. (b) Mixed-score attention. "Multi-Head Mixed-Score Attention" block in (a) consists of many independent copies of the mixed-score attentions arranged in parallel and fully connected (FC) layers at the input and output interfaces (not drawn).

GNN-based

1. **Learning Combinatorial Optimization Algorithms over Graphs**

- `Publication` : NeurIPS 2017
- `CO-problem` : MVC, MAXCUT, TSP
- `ML-type` : RL (Q-learning)
- `Component` : GNN (structure2vec, S2V)
- `Innovation` : This paper proposes a unique combination of reinforcement learning and graph embedding to address this challenge. The learned greedy policy behaves like a meta-algorithm that incrementally constructs a solution, and the action is determined by the output of a graph embedding network capturing the current state of the solution.
- `Link` : arXiv

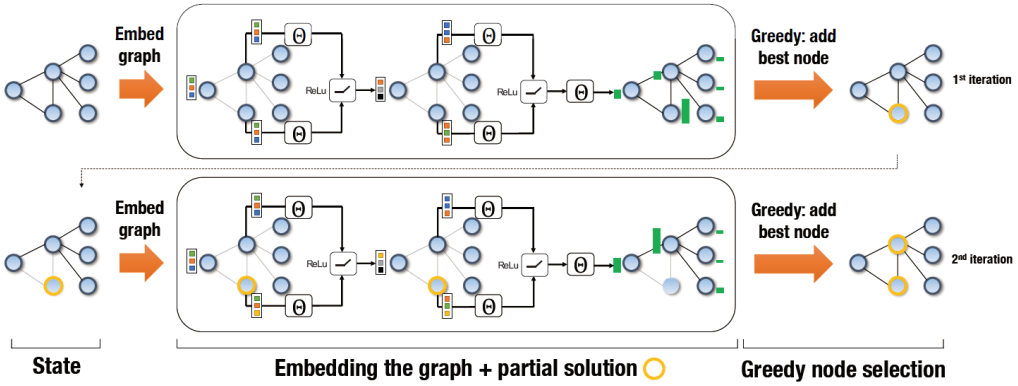
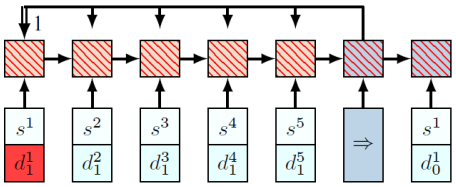
- 

Figure 1: Illustration of the proposed framework as applied to an instance of Minimum Vertex Cover. The middle part illustrates two iterations of the graph embedding, which results in node scores (green bars).

2. Reinforcement Learning for Solving the Vehicle Routing Problem

- `Publication` : NeurIPS 2018
- `CO-problem` : VRP
- `ML-type` : RL
- `Component` : GNN (GCN)
- `Innovation` : This paper presents an end-to-end framework for solving the Vehicle Routing Problem (VRP) using reinforcement learning.
- `Link` : arXiv

- 
Figure 1: Limitation of the Pointer Network. After a change in dynamic elements (d_1^1 in this example), the whole Pointer Network must be updated to compute the probabilities in the next decision point.

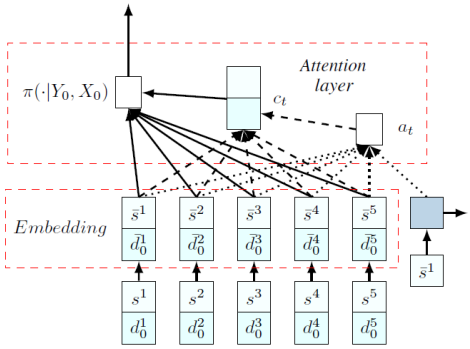


Figure 2: Our proposed model. The embedding layer maps the inputs to a high-dimensional vector space. On the right, an RNN decoder stores the information of the decoded sequence. Then, the RNN hidden state and embedded input produce a probability distribution over the next input using the attention mechanism.

3. Attention, Learn to Solve Routing Problems!

- `Publication` : NeurIPS 2018
- `CO-problem` : TSP, VRP
- `ML-type` : RL (REINFORCE+rollout baseline)
- `Component` : Attention, GNN
- `Innovation` : This paper proposes a model based on attention layers with benefits over the Pointer Network and we show how to train this model using REINFORCE with a simple baseline based on a deterministic greedy rollout, which we find is more efficient than using a value function.
- `Link` : arXiv

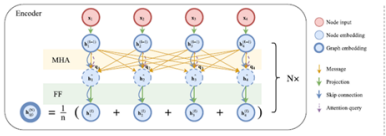
- 

Figure 1: Attention based encoder. Input nodes are embedded and processed by N sequential layers, each consisting of a multi-head attention (MHA) and node-wise feed-forward (FF) sub-layer. The graph embedding is computed as the mean of node embeddings. Best viewed in color.

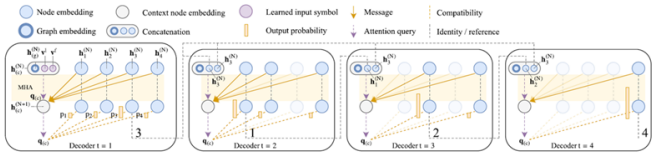
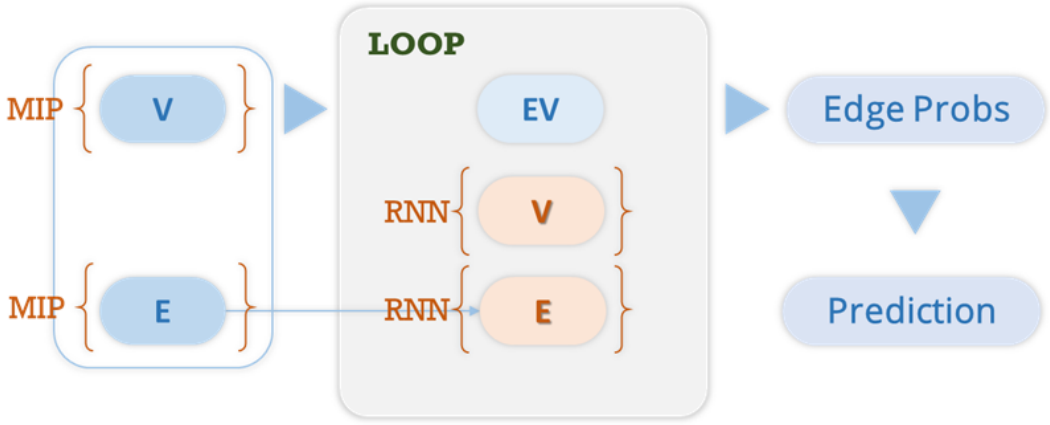
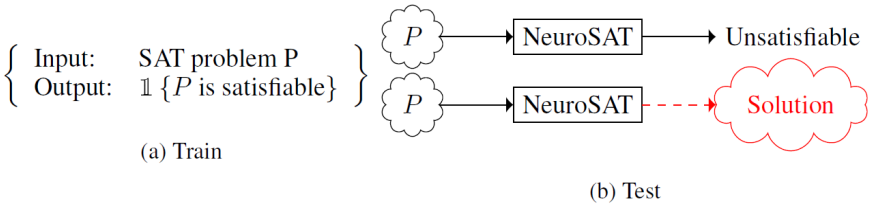


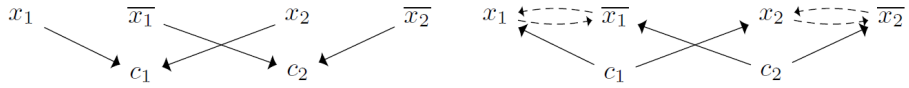
Figure 2: Attention based decoder for the TSP problem. The decoder takes as input the graph embedding and node embeddings. At each time step t, the context consist of the graph embedding and the embeddings of the first and last (previously output) node of the partial tour, where learned placeholders are used if $t = 1$. Nodes that cannot be visited (since they are already visited) are masked. The example shows how a tour $\pi = (3, 1, 2, 4)$ is constructed. Best viewed in color.

4. **Learning to Solve NP-Complete Problems - A Graph Neural Network for Decision TSP**
    - `Publication` : AAAI 2019
    - `CO-problem` : TSP
    - `ML-type` : SL
    - `Component` : RNN, GNN
    - `Innovation` : This paper proposes a GNN model where edges (embedded with their weights) communicate with vertices.
    - `Link` : arXiv
    - 

5. **Learning a SAT Solver from Single-Bit Supervision**
    - `Publication` : ICLR 2019
    - `CO-problem` : SAT
    - `ML-type` : SL
    - `Component` : GNN, LSTM
    - `Innovation` : NeuroSAT enforces both permutation invariance and negation invariance.
    - `Link` : arXiv
    - 

(a) Train (b) Test

Figure 1: We train NeuroSAT to predict whether SAT problems are satisfiable, providing only a single bit of supervision for each problem. At test time, when NeuroSAT predicts *satisfiable*, we can almost always extract a satisfying assignment from the network's activations. The problems at test time can also be substantially larger, more difficult, and even from entirely different domains than the problems seen during training.



- Figure 2: High-level illustration of NeuroSAT operating on the graph representation of $\{1|2, \overline{1}|\overline{2}\}$. On the top of both figures are nodes for each of the four literals, and on the bottom are nodes for each of the two clauses. At every time step t, we have an embedding for every literal and every clause. An iteration consists of two stages. First, each clause receives messages from its neighboring literals and updates it embedding accordingly (Figure 2a). Next, each literal receives messages from its neighboring clause as well as from its complement, and updates its embedding accordingly (Figure 2b).

6. End to end learning and optimization on graphs

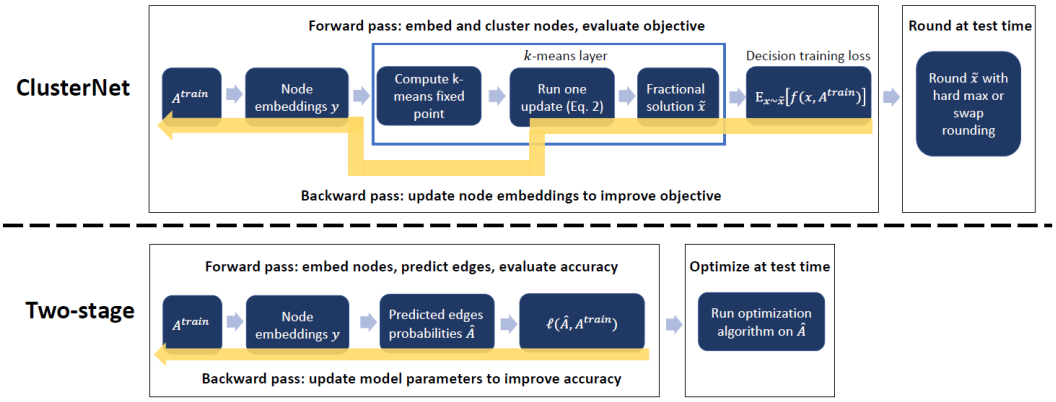
- `Publication` : NeurIPS 2019
- `CO-problem` : MAXCUT
- `ML-type` : UL
- `Component` : GNN
- `Innovation` : This paper proposed a new approach CLUSTERNET to this decision-focused learning problem: include a differentiable solver for a simple proxy to the true, difficult optimization problem and learn a representation that maps the difficult problem to the simpler one. (relax+differentiate)
- `Link` : arXiv
- 

Figure 1: Top: CLUSTERNET, our proposed system. Bottom: a typical two-stage approach.

7. Efficiently Solving the Practical Vehicle Routing Problem: A Novel Joint Learning Approach

- `Publication` : KDD 2020
- `CO-problem` : VRP
- `ML-type` : SL, RL (REINFORCE+rollout baseline)
- `Component` : GCN
- `Innovation` : GCN-NPEC model is based on the graph convolutional network (GCN) with node feature (coordination and demand) and edge feature (the real distance between nodes) as input and embedded. Separate decoders are proposed to decode the representations of these two features. The output of one decoder is the supervision of the other decoder. This paper also proposes a strategy that combines the

reinforcement learning manner with the supervised learning manner to train the model.

- `Link` : paper

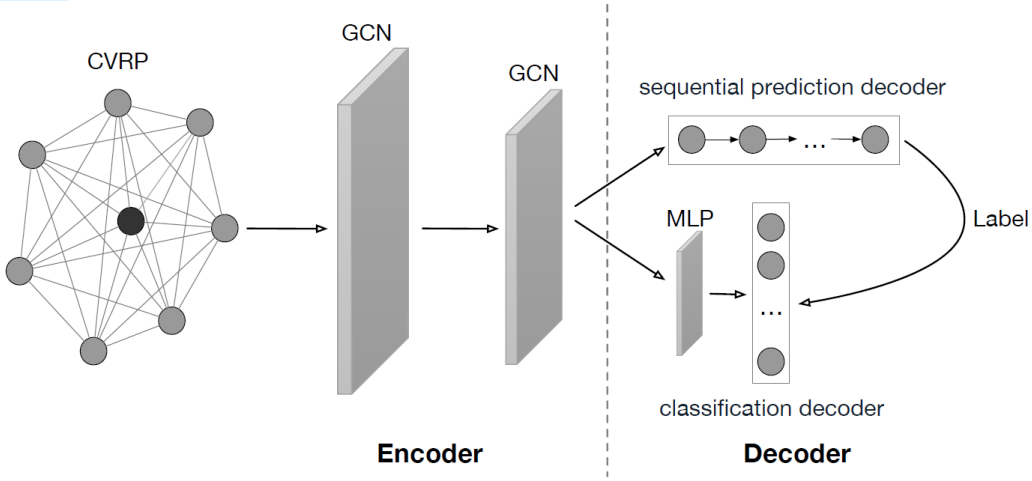
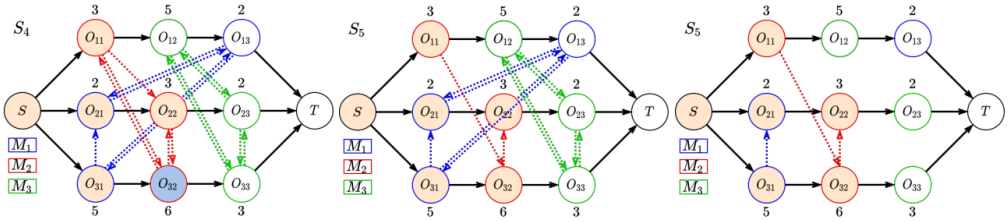


Figure 3: The overall architecture of our GCN-NPEC model.

8. **Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning**
   - `Publication` : NeurIPS 2020
   - `CO-problem` : JSSP
   - `ML-type` : RL (PPO)
   - `Component` : GNN
   - `Innovation` : This paper exploit the disjunctive graph representation of JSSP , and propose a Graph Neural Network based scheme to embed the states encountered during solving.
   - `Link` : arXiv



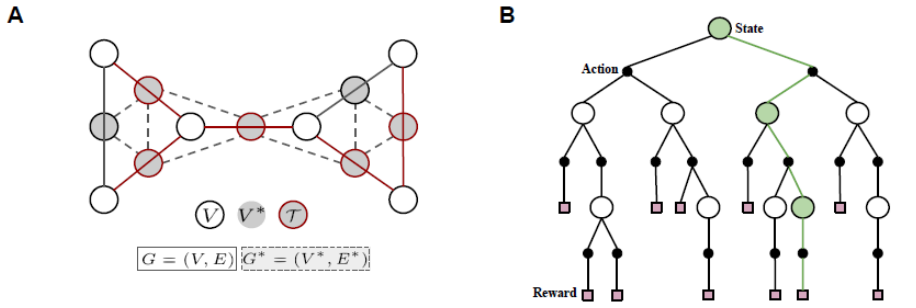
(a) Directed disjunctive graph (b) Removing-arc strategy (c) Adding-arc strategy

Figure 3: **Fully directed disjunctive graph, removing-arc strategy, and adding-arc strategy.** (a) is a fully directed disjunctive graph by replacing each undirected disjunctive arc with two opposite directed arcs. (b) shows the removing-arc strategy. The directed arc conflicting with the current decision is removed from the graph, i.e. arcs (O_{32}, O_{11}) and (O_{22}, O_{32}). (c) shows the adding-arc strategy. The directed arc following the current decision is added to the graph, i.e. arcs (O_{11}, O_{32}) and (O_{32}, O_{22}).

9. **Learning to Solve Combinatorial Optimization Problems on Real-World Graphs in Linear Time**
   - `Publication` : arXiv 2020
   - `CO-problem` : TSP, VRP
   - `ML-type` : RL
   - `Component` : GNN
   - `Innovation` : This paper develops a new framework to solve any combinatorial optimization problem over graphs that can be formulated as a single player game defined by states, actions, and rewards, including

minimum spanning tree, shortest paths, traveling salesman problem, and vehicle routing problem, problem, without expert knowledge.

- `Link` : arXiv



C

Problem	State	Action	Reward
MST	$G = (V, E), G^* = (V^*, E^*), W, \mathcal{T}$	$\mathcal{T} = \mathcal{T} \cup \{e\}$	$-\left(I(\mathcal{T}) + \sum_{e \in E_\pi} W(e)\right)$ (Eq. 4)
SSP	$G = (V, E), G^* = (V^*, E^*), W, \mathcal{Q}_i$	$\mathcal{Q}_i = \mathcal{Q}_i \cup \{e\}$	$-\sum_{i=1}^{\lvert V\rvert}\left(I(\mathcal{Q}_i) + \sum_{e \in \mathcal{Q}_i} W(e)\right)$
TSP	$G = (V, E), \bar{V} = \{\tau(1), .., \tau(i)\}$	$\bar{V} = \bar{V} \cup \{\tau(i+1)\}$	$-\sum_{i=1}^{\lvert V\rvert} \lVert \mathbf{v}_{\tau(i)} - \mathbf{v}_{\tau(i+1)} \rVert_2$ (Eq. 5)
VRP	$G = (V, E), \bar{V}_m = \{\tau(d), \tau_m(2), .., \tau_m(i)\}, M$	$\bar{V}_m = \bar{V}_m \cup \{\tau_m(i+1)\}$	$-\max_{m \in \{1,..,M\}} \left\{ \sum_{i \in \bar{V}_\tau(m)} \lVert \mathbf{v}_{\tau_m(i)} - \mathbf{v}_{\tau_m(i+1)} \rVert_2 \right\}$

Figure 1: Our unified framework: (A) The primal graph (white nodes and solid edges) and its edge-to-vertex line graph (gray nodes and dashed edges). Two nodes in the line graph are connected if the corresponding edges in the primal graph share a node. Notice that, while the number of primal edges (7) is equal to the number of dual nodes (7), the number of dual edges (10) is not necessarily equal to the number of primal nodes (6). (B) Combinatorial optimization as a single player game defined by states, actions, and reward. Traversing a path (green) from the root to a leaf node (pink square) corresponds to a solution for a problem. White nodes represent states and black nodes represent actions. From each state there may be many possible actions (more than the two illustrated here) representing the possible nodes or edges in the problem graph. The leaf nodes represent rewards or costs such as the sum of weights in MST or length of tour in TSP. (C) Graph algorithms for polynomial problems MST and SSP, and NP-hard problems TSP and VRP formulated as single player games by reinforcement learning using states, actions, and reward. For MST, the state includes the graph, line graph, weights, and selected edges $\mathcal{T}$ (red).

10. **Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning**
    - `Publication` : AAAI 2020 (Workshop)
    - `CO-problem` : TSP
    - `ML-type` : Hierarchical RL
    - `Component` : GNN
    - `Innovation` : GPNs build upon Pointer Networks by introducing a graph embedding layer on the input, which captures relationships between nodes. Furthermore, to approximate solutions to constrained combinatorial optimization problems such as the TSP with time windows, we train hierarchical GPNs (HGPNs) using RL, which learns a hierarchical policy to find an optimal city permutation under constraints.
    - `Link` : arXiv

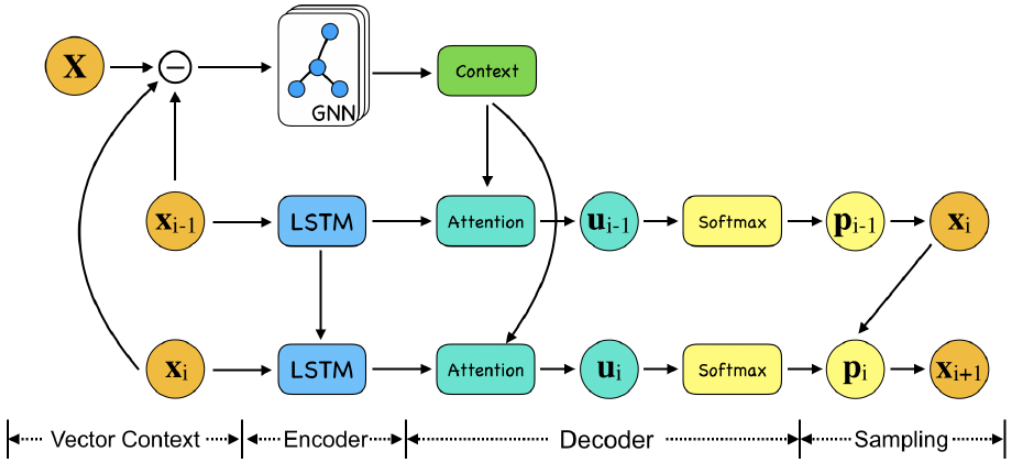
- 

Figure 2. Architecture for Graph Pointer Network. The current city coordinate $\mathbf{x}_i$ (we denote $\mathbf{x}_{\sigma(i)}$ as $\mathbf{x}_i$ for convenience) is encoded by the LSTM while $\bar{\mathbf{X}} = (\mathbf{X} - \mathbf{X}_i)$ is encoded as the vector context by a graph neural network. The encoded vectors are passed to the attention decoder, which outputs the pointer vector $\mathbf{u}_i$. The probability distribution over the next candidate city is $\mathbf{p}_i = \text{softmax}(\mathbf{u}_i)$. The next visited city $\mathbf{x}_{i+1}$ is sampled from $\mathbf{p}_i$.

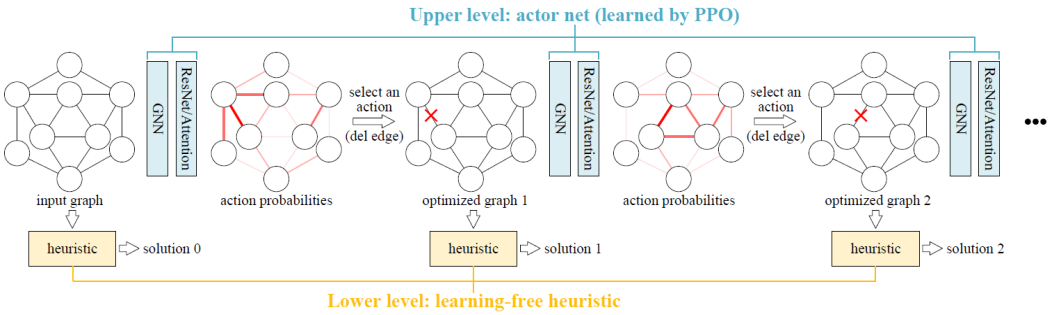
11. **A Bi-Level Framework for Learning to Solve Combinatorial Optimization on Graphs**
    - `Publication` : arXiv 2021
    - `CO-problem` : Directed Acyclic Graph scheduling, Graph Edit Distance, Hamiltonian Cycle Problem
    - `ML-type` : RL (PPO)
    - `Component` : GNN, ResNet, Attention
    - `Innovation` : This paper proposes a bi-level framework is developed with an upper-level learning method to optimize the graph (e.g. add, delete or modify edges in a graph), fused with a lower-level heuristic algorithm solving on the optimized graph. Such a bi-level approach simplifies the learning on the original hard CO and can effectively mitigate the demand for model capacity.
    - `Link` : arXiv
    - 

Figure 1: An overview of our bi-level hybrid MLCO solver. The graph structure is optimized at the upper level by an RL agent, and the optimized graphs are solved by heuristics at the lower level. The actions can be any modifications of the edges (i.e. adding, deleting edges and modifying edge attributes), and edge deletion is presented in this example.

12. **SOLO: Search Online, Learn Offline for Combinatorial Optimization Problems**
    - `Publication` : arXiv 2021

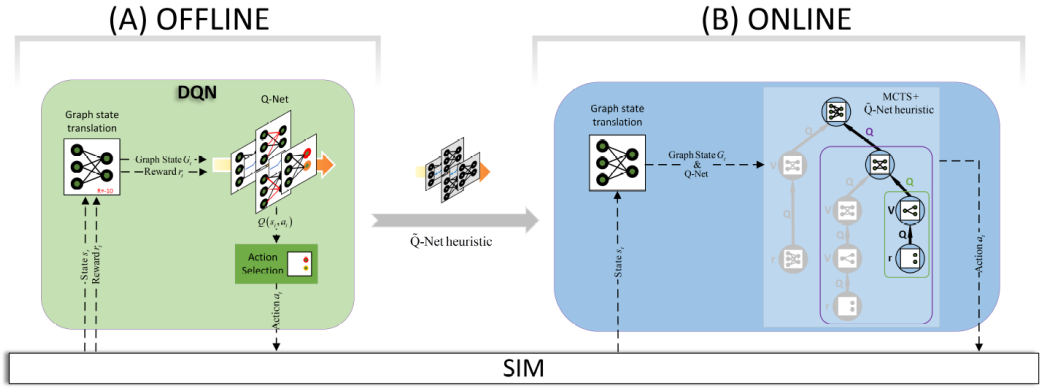
- `CO-problem` : VRP, JSSP
- `ML-type` : RL (DQN, MCTS)
- `Component` : GNN
- `Innovation` : Learn Offline -> DQN + Search Online -> MCTS
- `Link` : arXiv
- 

Figure 1: A schematic overview of SOLO. On the left, a depiction of our DQN training process, which produces the $\tilde{Q}$-Net heuristic. On the right is our planning procedure that, for each step, runs our modified MCTS with $\tilde{Q}$-Net as a heuristic.

Other

1. **Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning**
   - `Publication` : NeurIPS 2020
   - `CO-problem` : VRP
   - `ML-type` : RL
   - `Component` : /
   - `Innovation` : Policy evaluation with mixed-integer optimization. At the policy evaluation step, they formulate the action selection problem from each state as a mixed-integer program, in which they combine the combinatorial structure of the action space with the neural architecture of the value function by adapting the branch-and-cut approach.
   - `Link` : arXiv

Local Search

1. **Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search**
   - `Publication` : NeurIPS 2018
   - `CO-problem` : MIS, MVC, MC, SAT
   - `ML-type` : SL
   - `Component` : GNN (GCN)
   - `Innovation` : This paper proposes a model whose central component is a graph convolutional network that is trained to estimate the likelihood, for

each vertex in a graph, of whether this vertex is part of the optimal solution. The network is designed and trained to synthesize a diverse set of solutions, which enables rapid exploration of the solution space via tree search.

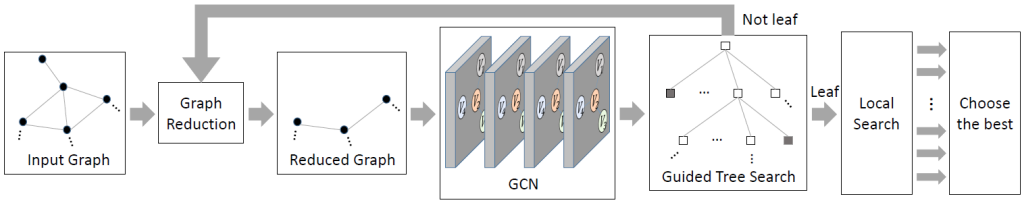
- `Link` : arXiv
- 

Figure 1: Algorithm overview. First, the input graph is reduced to an equivalent smaller graph. Then it is fed into the graph convolutional network f, which generates multiple probability maps that encode the likelihood of each vertex being in the optimal solution. The probability maps are used to iteratively label the vertices until all vertices are labelled. A complete labelling corresponds to a leaf in the search tree. Internal nodes in the search tree represent incomplete labellings that are generated along the way. The complete labellings generated by the tree search are refined by rapid local search. The best result is used as the final output.

2. **Learning to Perform Local Rewriting for Combinatorial Optimization**

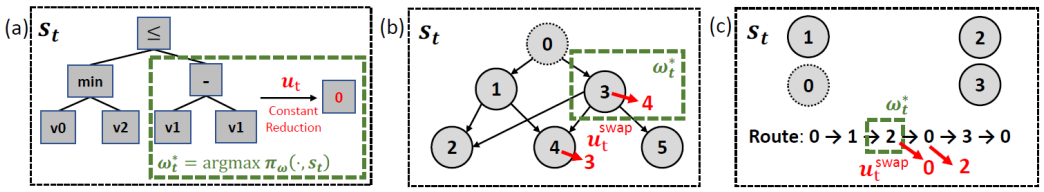
- `Publication` : NeurIPS 2019
- `CO-problem` : JSSP, VRP
- `ML-type` : RL (Actor-Critic)
- `Component` : LSTM
- `Innovation` : NeuRewriter learns a policy to pick heuristics, and rewrite local components of the current solution to iteratively improve it until convergence. The policy factorizes into a region-picking and a rule-picking component, each of which parameterized by an NN trained with actor-critic in RL.
- `Link` : arXiv
- 

Figure 2: The instantiation of `NeuRewriter` for different domains: **(a)** expression simplification; **(b)** job scheduling; and **(c)** vehicle routing. In **(a)**, s_t is the expression parse tree, where each square represents a node in the tree. The set $\Omega(s_t)$ includes every sub-tree rooted at a non-terminal node, from which the region-picking policy selects $\omega_t \sim \pi_\omega(\omega_t|s_t))$ to rewrite. Afterwards, the rule-picking policy predicts a rewriting rule $u_t \in \mathcal{U}$, then rewrites the sub-tree ω_t to get the new tree s_{t+1}. In **(b)**, s_t is the dependency graph representation of the job schedule. Each circle with index greater than 0 represents a job node, and node 0 is an additional one representing the machine. Edges in the graph reflect job dependencies. The region-picking policy selects a job ω_t to re-schedule from all job nodes, then the rule-picking policy chooses a moving action u_t for ω_t, then modifies s_t to get a new dependency graph s_{t+1}. In **(c)**, s_t is the current route, and ω_t is the node selected to change the visit order. Node 0 is the depot, and other nodes are customers with certain resource demands. The region-picking policy and the rule-picking policy work similarly to the job scheduling ones.

3. **A Learning-based Iterative Method for Solving Vehicle Routing Problems**

- `Publication` : ICLR 2020
- `CO-problem` : VRP
- `ML-type` : RL (REINFORCE)
- `Component` : /
- `Innovation` : "Learn to Improve" (L2I)

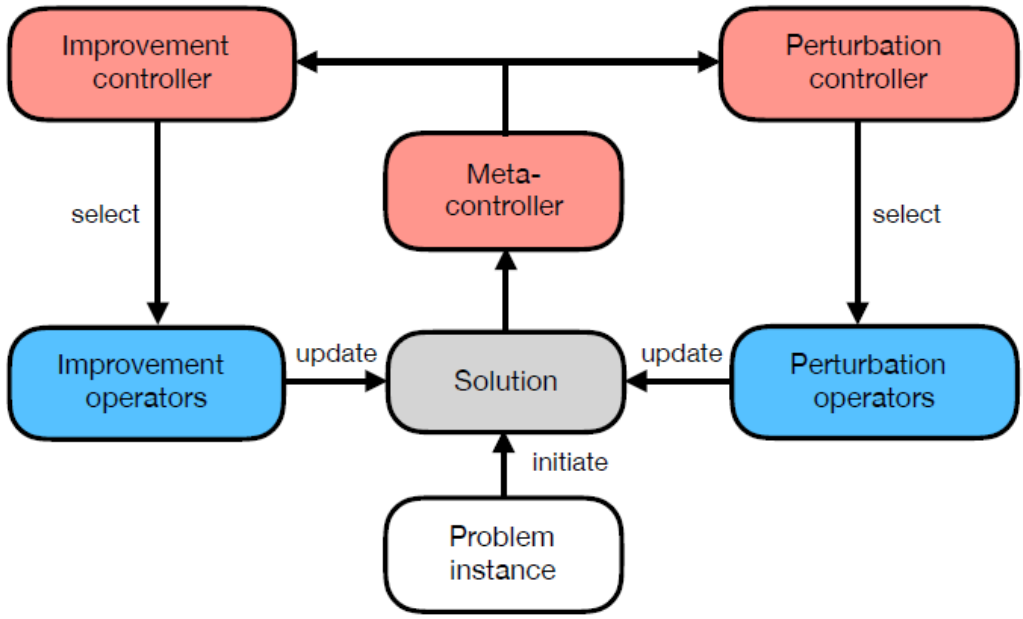
    - hierarchical framework: separate heuristic operators into two classes, namely improvement operators and perturbation operators. At each state, we choose the class first and then choose operators within the class. Learning from the current solution is made easier by focusing RL on the improvement operators only.
    - ensemble method: trains several RL policies at the same time, but with different state input features.
  - `Link` : paper
  - 

Figure 2: Our hierarchy framework. Given a problem instance, our algorithm first generates a feasible solution. Then it iteratively updates the solution with an improvement operator selected by an RL-based controller or with a perturbation operator chosen by a rule-based controller. After a certain number of steps, we choose the best one among all visited solutions.

4. **Exploratory Combinatorial Optimization with Reinforcement Learning**
   - `Publication` : AAAI 2020
   - `CO-problem` : MAXCUT
   - `ML-type` : RL (DQN)
   - `Component` : GNN (MPNN)
   - `Innovation` : ECO-DQN combines S2V-DQN, Reversible Actions, Observation Tuning and Intermediate Rewards.
   - `Link` : arXiv
5. **Rewriting By Generating: Learn To Solve Large-Scale Vehicle Routing Problems**
   - `Publication` : ICLR 2021
   - `CO-problem` : VRP

- `ML-type` : Hierarchical RL (REINFORCE)
- `Component` : LSTM, K-means, PCA
- `Innovation` : Inspired by the classical idea of Divide-and-Conquer, this paper presents a novel Rewriting-by-Generating(RBG) framework with hierarchical RL agents to solve large-scale VRPs. RBG consists of a rewriter agent that refines the customer division globally and an elementary generator to infer regional solutions locally.
- `Link` : paper

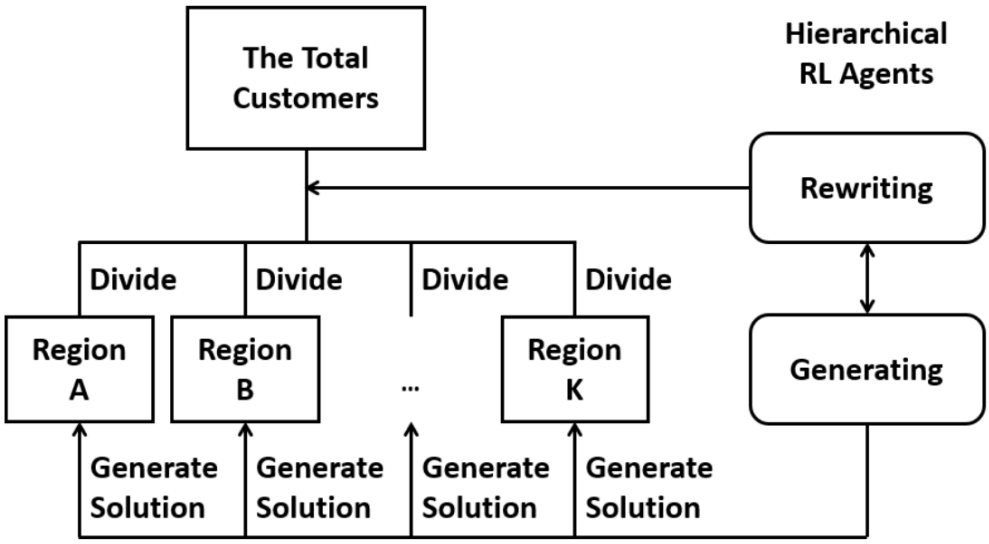


Figure 1: The overview of the Rewriting-by-Generating.

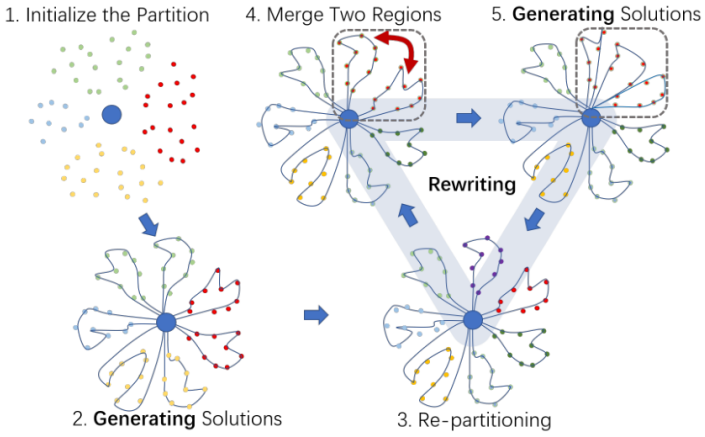
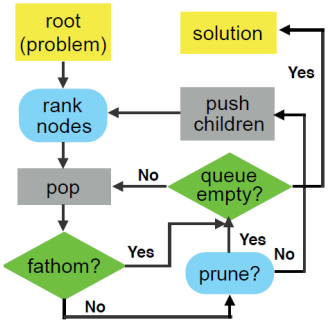


Figure 2: The core idea behind our Rewriting-by-Generating framework, where nodes denote customers with color representing regions, and solid lines denote routes.

B&B-based

1. **Learning to Search in Branch and Bound Algorithms**
   - `Publication` : NeurIPS 2014
   - `CO-problem` : MILP
   - `ML-type` : RL (Imitation learning)
   - `Component` : /

- `Innovation` : This paper proposed an approach that learns branch-and-bound by imitation learning. (node selection policy and node pruning policy)
- `Link` : paper
- 

Algorithm 1 Policy Learning (π_S^*, π_P^*)

$\pi_P^{(1)} = \pi_P^*, \pi_S^{(1)} = \pi_S^*, \mathcal{D}_S = \{\}, \mathcal{D}_P = \{\}$
for $k = 1$ **to** N **do**
  for Q **in** problem set $\mathcal{Q}$ **do**
    $\mathcal{D}_S^{(Q)}, \mathcal{D}_P^{(Q)} \leftarrow \text{CollectExample}(Q, \pi_P^{(k)}, \pi_S^{(k)})$
    $\mathcal{D}_S \leftarrow \mathcal{D}_S \cup \mathcal{D}_S^{(Q)}, \quad \mathcal{D}_P \leftarrow \mathcal{D}_P \cup \mathcal{D}_P^{(Q)}$
  $\pi_S^{(k+1)}, \pi_P^{(k+1)} \leftarrow$ train classifiers using $\mathcal{D}_S$ and $\mathcal{D}_P$
return Best $\pi_S^{(k)}, \pi_P^{(k)}$ on dev set

Figure 2: **Our method at runtime (left) and the policy learning algorithm (right).** *Left*: our policy-guided branch-and-bound search. Procedures in the rounded rectangles (shown in blue) are executed by policies. *Right*: the DAgger learning algorithm. We start by using oracle policies π_S^* and π_P^* to solve problems in $\mathcal{Q}$ and collect examples along oracle trajectories. In each iteration, we retrain our policies on all examples collected so far (training sets $\mathcal{D}_D$ and $\mathcal{D}_S$), then collect additional examples by running the newly learned policies. The CollectExample procedure is described in Algorithm 2.

2. **Learning to Branch in Mixed Integer Programming**
   - `Publication` : AAAI 2016
   - `CO-problem` : MIP
   - `ML-type` : RL (Imitation Learning)
   - `Component` : /
   - `Innovation` : This paper proposes the first successful ML framework for variable selection in MIP.
   - `Link` : paper
3. **Learning to Run Heuristics in Tree Search**
   - `Publication` : IJCAI 2017
   - `CO-problem` : MIP
   - `ML-type` : SL
   - `Component` : classifier
   - `Innovation` : Central to this approach is the use of Machine Learning (ML) for predicting whether a heuristic will succeed at a given node.
   - `Link` : arXiv
4. **Improving Optimization Bounds using Machine Learning: Decision Diagrams meet Deep Reinforcement Learning**
   - `Publication` : AAAI 2019
   - `CO-problem` : MIS, MAXCUT
   - `ML-type` : RL (Q-learning)
   - `Component` : /
   - `Innovation` : This paper proposes an innovative and generic approach based on deep reinforcement learning for obtaining an ordering for tightening the bounds obtained with relaxed and restricted DDs (decision diagrams).

- `Link` : arXiv

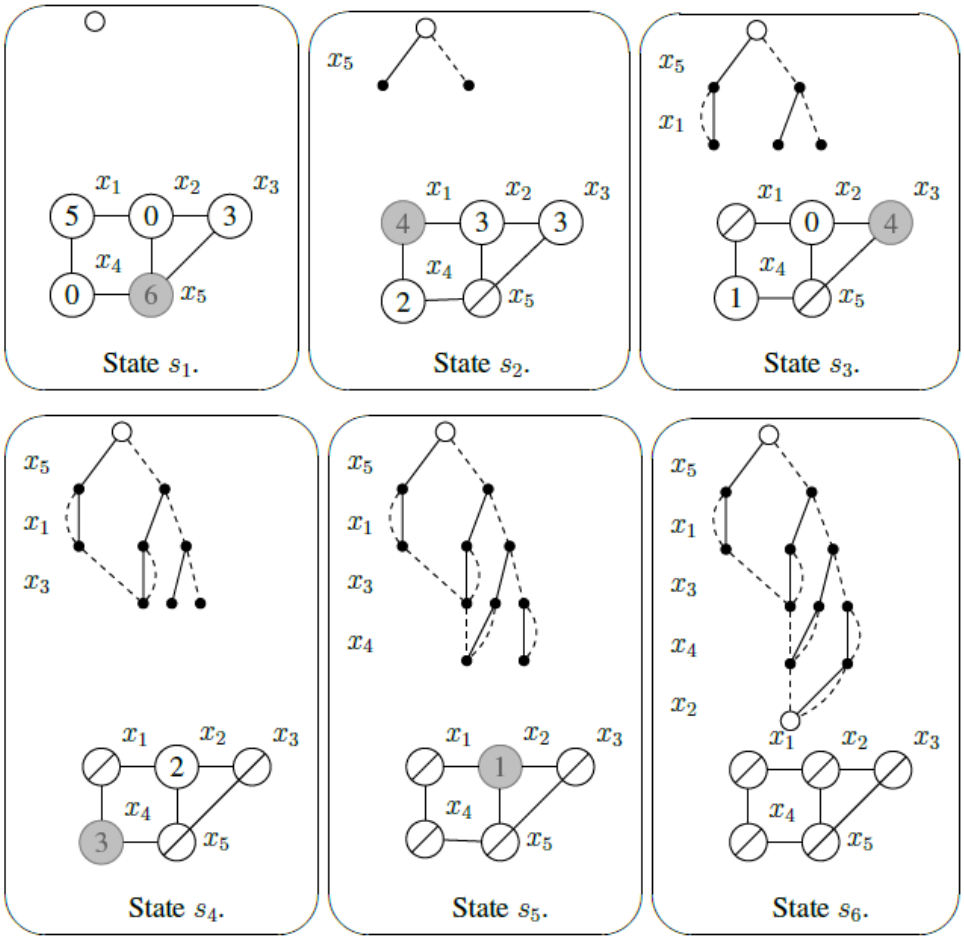


Figure 1: Example of an exact DD construction for a MISP instance, following policy $\pi = \text{argmax}_a Q^\pi(s, a)$.

5. **Exact Combinatorial Optimization with Graph Convolutional Neural Networks**
   - `Publication` : NeurIPS 2019
   - `CO-problem` : MIIP
   - `ML-type` : RL (Imitation learning)
   - `Component` : GNN (GCN)
   - `Innovation` : This paper proposes a new graph convolutional neural network model for learning branch-and-bound variable selection policies, which leverages the natural variable-constraint bipartite graph representation of mixed-integer linear programs.
   - `Link` : arXiv

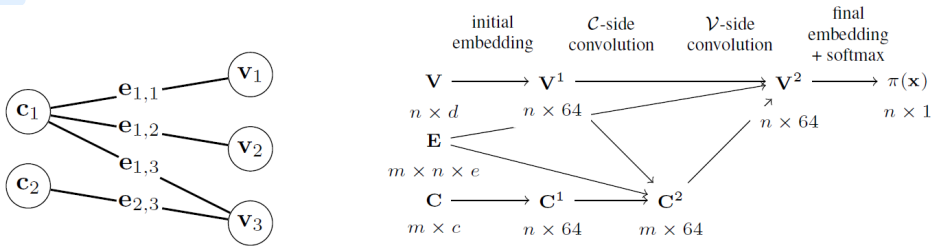


Figure 2: Left: our bipartite state representation $\mathbf{s}_t = (\mathcal{G}, \mathbf{C}, \mathbf{E}, \mathbf{V})$ with $n = 3$ variables and $m = 2$ constraints. Right: our bipartite GCNN architecture for parametrizing our policy $\pi_\theta(\mathbf{a} \mid \mathbf{s}_t)$.

6. **Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization**

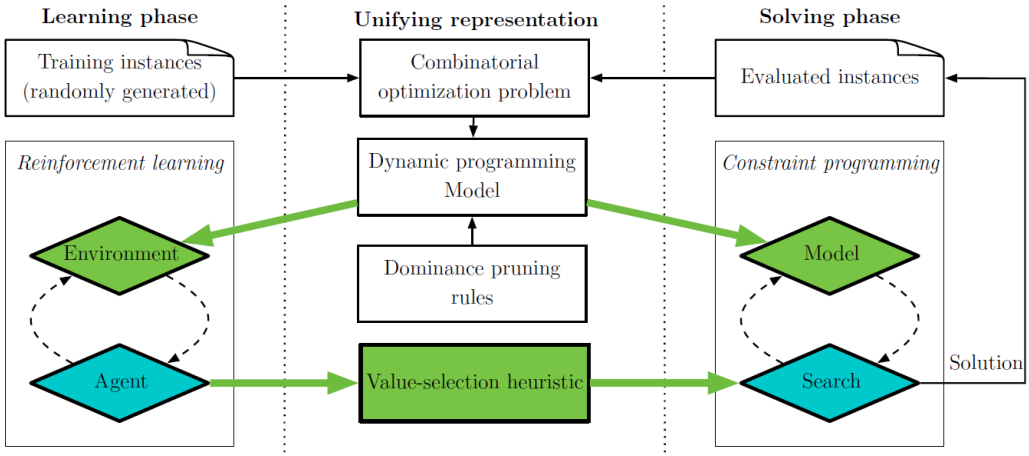
- `Publication` : AAAI 2021
- `CO-problem` : TSP, Portfolio Optimization Problem
- `ML-type` : RL (DQN, PPO)
- `Component` : GNN (GAT), Set Transformer
- `Innovation` : This paper propose a general and hybrid approach, based on DRL and CP (Constraint Programming), for solving combinatorial optimization problems. The core of this approach is based on a dynamic programming formulation, that acts as a bridge between both techniques.
- `Link` : arXiv
- 

Figure 1: Overview of our framework for solving COPs.

7. **Solving Mixed Integer Programs Using Neural Networks**
   - `Publication` : arXiv 2021
   - `CO-problem` : MIP
   - `ML-type` : RL (Imitation Learning)
   - `Component` : GNN (Bipartite graph)
   - `Innovation` : Neural Diving + Neural Branching
     - Neural Diving learns a deep neural network to generate multiple partial assignments for its integer variables, and the resulting smaller MIPs for un-assigned variables are solved with SCIP to construct high quality joint assignments.
     - Neural Branching learns a deep neural network to make variable selection decisions in branch-and-bound to bound the objective value gap with a small tree.
   - `Link` : arXiv
   -

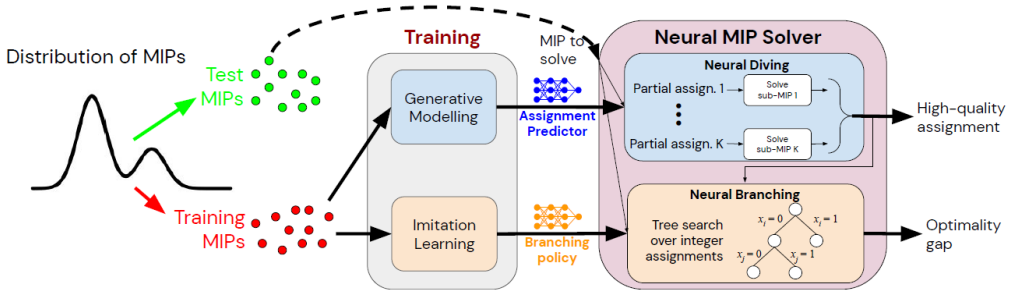


Figure 1 Our approach constructs two neural network-based components to use in a MIP solver, *Neural Diving* and *Neural Branching*, and combine them to produce a *Neural Solver* customized to a given MIP dataset.

Appendix

Template

---

1. - `Publication` :
   - `CO-problem` :
   - `ML-type` :
   - `Component` :
   - `Innovation` :
   - `Link` : arXiv

   -

---

2. - `Publication` :
   - `CO-problem` :
   - `Link` : arXiv

To Be Classified

1. **Reinforcement Learning for (Mixed) Integer Programming: Smart Feasibility Pump**
   - `Publication` : ICML 2021 (Workshop)
   - `CO-problem` : MIP
   - `Link` : arXiv
2. **Learning Local Search Heuristics for Boolean Satisfiability**
   - `Publication` : NeurIPS 2019
   - `CO-problem` : SAT
   - `Link` : paper
3. **Reinforcement Learning on Job Shop Scheduling Problems Using Graph Networks**

- `Publication` : arXiv 2020
- `CO-problem` : JSSP
- `Link` : arXiv

4. **A Generalized Reinforcement Learning Algorithm for Online 3D Bin-Packing**
   - `Publication` : AAAI 2020 (Workshop)
   - `CO-problem` : 3D-BPP
   - `Link` : arXiv
5. **A Reinforcement Learning Approach to Job-shop Scheduling**
   - `Publication` : IJCAI 2020
   - `CO-problem` : JSSP
   - `Link` : paper
6. **A Reinforcement Learning Environment For Job-Shop Scheduling**
   - `Publication` : arXiv 2021
   - `CO-problem` : JSSP
   - `Link` : arXiv
7. **Improving Optimization Bounds Using Machine Learning ~ Decision Diagrams Meet Deep Reinforcement Learning**
   - `Publication` : AAAI 2019
   - `CO-problem` : MIS, MAXCUT
   - `Link` : arXiv
8. **Online 3D Bin Packing with Constrained Deep Reinforcement Learning**
   - `Publication` : AAAI 2021
   - `CO-problem` : 3D-BPP
   - `Link` : arXiv
9. **A Data-Driven Approach for Multi-level Packing Problems in Manufacturing Industry**
   - `Publication` : KDD 2019
   - `CO-problem` : 3D-BPP
   - `Link` : ACM DL
10. **Solving a New 3D Bin Packing Problem with Deep Reinforcement Learning Method**
    - `Publication` : arXiv 2017
    - `CO-problem` : 3D-BPP
    - `Link` : arXiv
11. **Meta-Learning-based Deep Reinforcement Learning for Multiobjective Optimization Problems**
    - `Publication` : arXiv 2017
    - `CO-problem` : TSP
    - `Link` : arXiv
12. **Dynamic Job-Shop Scheduling Using Reinforcement Learning Agents**

- `Publication` : ROBOT AUTON SYST 2000
- `CO-problem` : JSSP
- `Link` : paper