Machine Learning for Solving Large-scale Integer Programming Problems

Zaiwen Wen

Beijing International Center For Mathematical Research
Peking University

Outline

Maxcut: 0.878 bounds

- For graph (V, E) and weights $w_{ij} = w_{ji} \geq 0$, the maxcut problem is

$$(Q) \quad \max_{x} \sum_{i<j} w_{ij}(1 - x_i x_j), \ \text{ s.t. } \ x_i \in \{-1, 1\}$$

- SDP relaxation

$$(SDP) \quad \max_{X \in S^n} \sum_{i<j} w_{ij}(1 - X_{ij}), \ \text{ s.t. } \ X_{ii} = 1, X \succeq 0$$

Compute the decomposition $X = V^\top V$, where $V = [v_1, v_2, \ldots, v_n]$
- Rounding: generate a vector r uniformly distributed on the unit sphere, i.e., $\|r\|_2 = 1$, set

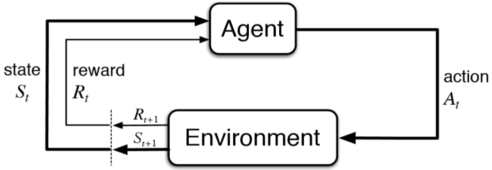
$$x_i = \begin{cases} 1 & v_i^\top r \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

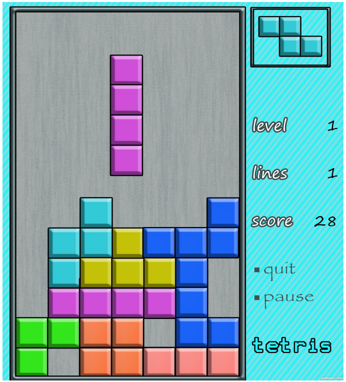
- Let $Z^*_{(SDP)}$ and $Z^*_{(Q)}$ be the optimal values of (SDP) and (Q)

$$E(W) \geq 0.878 Z^*_{(SDP)} \geq 0.878 Z^*_{(Q)}$$

Reinforcement Learning

Consider an infinite-horizon discounted Markov decision process (MDP), usually defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \rho_0, \gamma)$;





- The policy is supposed to maximize the total expected reward:

$$\max_{\pi} \mathrm{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right], \text{ with } s_0 \sim \rho_0, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t, a_t).$$

Erdos Goes Neural

- The probability distribution $\mathcal{D}$ in *Erdos* is learned by a GNN.
- A "good" probability distribution leads to higher quality solutions.

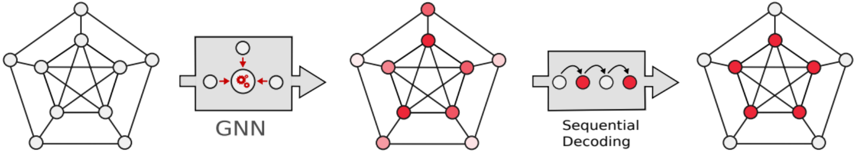


Figure: Illustration of the "Erdos goes neural" pipeline.

- Optimization on explicit formulation of the expectation.
- Maximum clique problem:

$$\ell(\mathcal{D}) = \gamma - (\beta + 1) \sum_{(v_i, v_j) \in E} w_{ij} p_i p_j + \frac{\beta}{2} \sum_{v_i \neq v_j} p_i p_j .$$

Parameterized Probabilistic Model

- **MCPG**: construct a parameterized model with parameter θ to output p_θ and generate $x \sim p_\theta$ by Monte Carlo sampling

- MCPG: optimization over the probabilistic space.
- Erdos: optimization on the expectation of objective function.

Outline

Binary Optimization

Let f be arbitrary (even non-smooth) cost function:

$$\min \quad f(x), \qquad \text{s.t.} \quad x \in \mathcal{B}_n = \{-1, 1\}^n.$$

- Example: maxcut problem on $G = (V, E)$

$$\max \quad \sum_{(i,j)\in E} w_{ij}(1 - x_i x_j), \qquad \text{s.t.} \quad x \in \{-1, 1\}^n.$$

- Example: maxSAT problem:

$$\begin{aligned} \max_{x\in\{-1,1\}^n} \quad & \sum_{c^i\in C_1} \max\{c_1^i x_1, c_2^i x_2, \cdots, c_n^i x_n, 0\}, \\ \text{s.t.} \quad & \max\{c_1^i x_1, c_2^i x_2, \cdots, c_n^i x_n, 0\} = 1, \quad \text{for } c^i \in C_2 \end{aligned}$$

- Binary optimization is NP-hard due to the combinatorial structure.

Probabilistic Approach

Let $\mathcal{X}^*$ be the set of optimal solutions and consider the distribution,

$$q^*(x) = \frac{1}{|\mathcal{X}^*|}\mathbf{1}_{\mathcal{X}^*}(x) = \begin{cases} \frac{1}{|\mathcal{X}^*|}, & x \in \mathcal{X}^*, \\ 0, & x \notin \mathcal{X}^*. \end{cases}$$

Motivation: Searching for optimal points $\mathcal{X}^* \Rightarrow$ Constructing a distribution $p_\theta(x)$ converging to $q^*(x)$.

- A universal approach for various binary optimization problems.
- Algorithms for continuous optimization can be applied.
- The optimal points set $\mathcal{X}^*$ is unknown.

Gibbs distributions

- To approximate q^*, we introduce Gibbs distributions,

$$q_\lambda(x) = \frac{1}{Z_\lambda} \exp\left(-\frac{f(x)}{\lambda}\right), \quad x \in \mathcal{B}_n,$$

  where $Z_\lambda = \sum_{x \in \mathcal{B}_n} \exp\left(-\frac{f(x)}{\lambda}\right)$ is the normalizer.
- Given the optimal objective value f^*, for any $x \in \mathcal{B}_n$,

$$\begin{aligned} q_\lambda(x) = \frac{\exp\left(\frac{f^* - f(x)}{\lambda}\right)}{\sum_{x \in \mathcal{B}_n} \exp\left(\frac{f^* - f(x)}{\lambda}\right)} &= \frac{\exp\left(\frac{f^* - f(x)}{\lambda}\right)}{|\mathcal{X}^*| + \sum_{x \in \mathcal{B}_n / \mathcal{X}^*} \exp\left(\frac{f^* - f(x)}{\lambda}\right)} \\ &\to \frac{1}{|\mathcal{X}^*|} \mathbf{1}_{\mathcal{X}^*}(x) = q^*, \quad \text{as } \lambda \to 0. \end{aligned}$$

- The calculation of q_λ does not require knowledge of $\mathcal{X}^*$.

Parameterized Probabilistic Model

- KL divergence:

$$\mathrm{KL}\left(p_\theta \,\|\, q_\lambda\right) = \sum_{x \in \mathbf{B}_n} p_\theta(x) \log \frac{p_\theta(x)}{q_\lambda(x)}.$$

- In order to reduce the discrepancy between p_θ and q_λ, the KL divergence is supposed to be minimized:

$$\begin{aligned}\mathrm{KL}\left(p_\theta \,\|\, q_\lambda\right) &= \frac{1}{\lambda} \sum_{x \in \mathbf{B}_n} p_\theta(x) f(x) + \sum_{x \in \mathbf{B}_n} p_\theta(x) \log p_\theta(x) + \log Z_\lambda \\ &= \frac{1}{\lambda}\left(\mathbb{E}_{p_\theta}\left[f(x)\right] + \lambda \mathbb{E}_{p_\theta}\left[\log p_\theta(x)\right]\right) + \log Z_\lambda.\end{aligned}$$

- Loss Function (Z_λ is a constant):

$$\boxed{\min_\theta \quad L_\lambda(\theta) = \mathbb{E}_{p_\theta}\left[f(x)\right] + \lambda \mathbb{E}_{p_\theta}\left[\log p_\theta(x)\right]}$$

Gradient for the Loss Function

Lemma 1

Suppose for any $x \in \mathcal{B}_n$, $p_\theta(x)$ is differentiable with respect to θ. For any constant $c \in \mathrm{Re}$, we denote the advantage function

$$A_\lambda(x;\theta,c) := f(x) + \lambda \log p_\theta(x) - c.$$

Then, the gradient of the loss function is given by

$$\nabla_\theta L_\lambda(\theta) = \mathbb{E}_{p_\theta}\left[A_\lambda(x;\theta,c)\nabla_\theta \log p_\theta(x)\right].$$

One candidate for c is

$$c = \mathbb{E}_{p_\theta}[f(x)].$$

Very similar to the policy gradient in reinforcement learning!

Extension: general constrained problem

- Consider

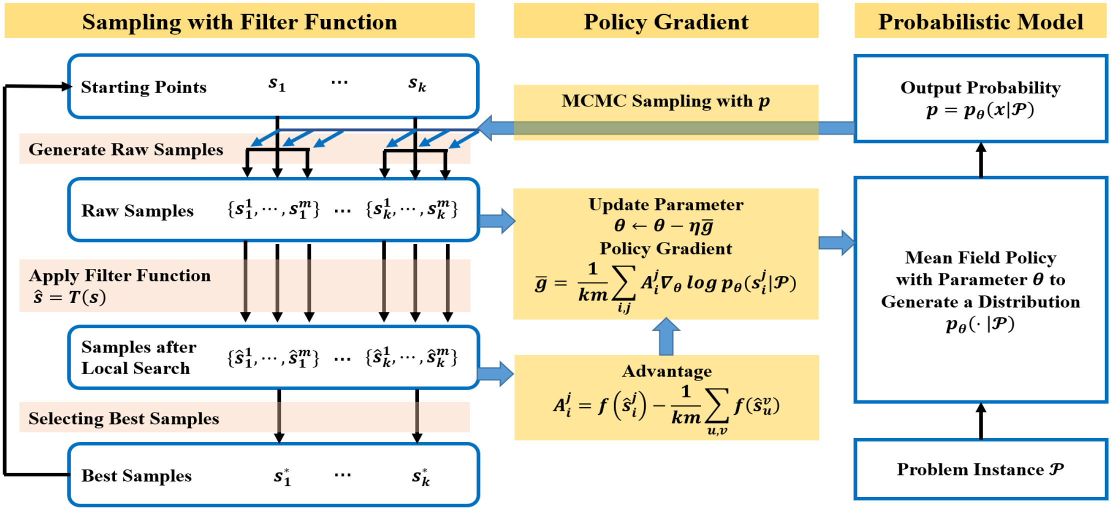
$$x^* = \arg\min_{x} \quad f(x), \ \text{ s.t. } \ c(x) = 0, \ x \in \mathcal{B}_n$$

- L1 exact penalty problem

$$x_\sigma^* = \arg\min_{x \in \mathcal{B}_n} f_\sigma(x) := f(x) + \sigma \|c(x)\|_1$$

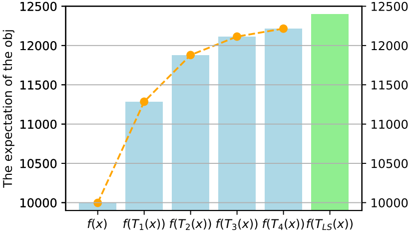
- Let $\varpi := \min_{x \in \mathcal{B}_n} \{\|c(x)\|_1 \mid \|c(x)\|_1 \neq 0\}$ and $f^* = \min_{x \in \mathcal{B}_n} f(x)$. Define $\bar{\sigma} = (f_\sigma(x^*) - f^*)/\varpi \geq 0$.
- For all $\sigma \geq \bar{\sigma}$, x^* is a global minima of the penalty problem and x_σ^* is also a global minima of the constrained problem.

Pipeline of MCPG

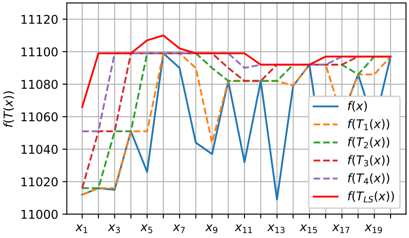


Filter Function

- The filter function T projects x to a better one in the neighborhood.
- Applied with the filter function, $f(T(x))$ has fewer local minima and the same global minimum as the original one.



(a) Expectation of the objective function.



(b) A selected sequence of solutions.

Filter Function

Definition 2 (Filter Function)

For each $x \in \mathcal{B}_n$, let $\mathcal{N}(x) \subset \mathcal{B}_n$ be a neighborhood of x such that $x \in \mathcal{N}(x)$, $|\mathcal{N}(x)| \geq 2$ and any point in $\mathcal{N}(x)$ can be reached by applying a series of “simple" operations to x. A filter function $T(x)$ is defined as

$$T(x) \in \arg\min_{\hat{x} \in \mathcal{N}(x)} f(\hat{x}),$$

where $T(x)$ is arbitrarily chosen if there exists multiple solutions.

- Projection to the best solution on the neighborhood:

$$T_k(x) = \arg\min_{\|\hat{x}-x\|_1 \leq 2k} f(\hat{x}), \quad \mathcal{N}(x) = \{\hat{x} \mid \|\hat{x} - x\|_1 \leq 2k\}.$$

- Algorithms serves as the filter function:

$$T_{LS}(x) = \texttt{LocalSearch}_f(x).$$

Local Search

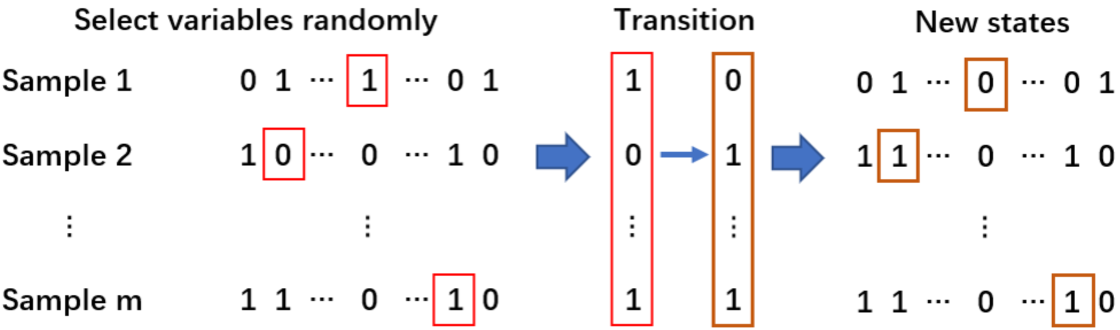
Local Search:

- **Generality**: Local search works for various kinds of problem.
- **Efficiency**: GPUs allow parallel access to the same indexed variable for a large number of samples.

Pipeline of Local Search with flipping operation:

1. Choose a single variable from the current solution x.
2. Flip the variable to its opposite value.
3. Evaluate the new solution to determine if it is improvement.
4. If it is, the variable is flipped to its opposite value
5. Back to Step 1 and continues to the next index in I.

Large-Scale Parallel Sampling on GPU



- GPU: quick for parallel accessing but slow for memory copying.

Sampling in MCPG

- constructs large number of short chains,
- discards all previous states in transition (no memory copying),
- outputs the last states for all chains.

Probabilistic Model Applied with Filter Function

- MCPG focuses on the following modified binary optimization:

$$\min \quad f(T(x)), \qquad \textbf{s.t.} \qquad x \in \mathcal{B}_n.$$

- The probabilistic model is equivalent to

$$\min_{\theta} \quad L_\lambda(\theta; \mathcal{P}) = \mathbb{E}_{p_\theta}\left[f(T(x))\right] + \lambda \mathbb{E}_{p_\theta}\left[\log p_\theta(x|\mathcal{P})\right].$$

- Empirical gradient:

$$\bar{g}_\lambda(\theta) = \frac{1}{|S|}\sum_{x \in S} A_\lambda(x;\theta) \nabla_\theta \log p(x|\theta;\mathcal{P}).$$

where S is the sample set extracted from distribution $p_\theta(\cdot|\mathcal{P})$ and

$$A_\lambda(x;\theta) := f(T(x)) + \lambda \log p_\theta(x|\mathcal{P}) - \frac{1}{|S|}\sum_{x \in S} f(T(x)).$$

Binary Optimization and Probabilistic model

For an arbitrary function f on $\mathcal{B}_n$, we define the B as

$$\mathcal{G}(f) = \min_{x \in \mathcal{B}_n \setminus \mathcal{X}^*} f(x) - f^*. \tag{1}$$

Proposition 1

For any $0 < \delta < 1$*, suppose* $L_\lambda(\theta) - f^* < (1-\delta)\mathcal{G}(f)$*, then*

$$\mathbb{P}(x \in \mathcal{X}^*) > \delta.$$

Therefore, for $x^1, \ldots, x^m$ *independently sampled from* p_θ, $\min_k f(x^k) = f^*$ *with probability at least* $1-(1-\delta)^m$.

The above proposition shows that with a optimized probabilistic model, the obtained probability from the optimal solutions is linearly dependent on the gap between the expectation and the minimum of f.

Impact of the Filter Function

- When $T(x) = x$, it means that x is a local minimum point.
- For any given $x \in \mathcal{B}_n$, there exists a corresponding local minimum point by applying the filter function T to x for many times.
- We can divide the set $\mathcal{B}_n$ into subsets with respect to the classification of local minima.

Let $X_1, X_2, ..., X_r$ be a partition of $\mathcal{B}_n$ such that for any $j \in \{1, \ldots, r\}$, every $x \in X_j$ has the same corresponding local minimum point.

Proposition 2

If there exists some $x \in \mathcal{B}_n$ such that $p_\theta(x) > 0$ and $f(x) > f(T(x))$, then for any sufficiently small $\lambda > 0$ satisfying

$$\mathbb{E}_{p_\theta}[f(x) - f(T(x))] \geq \lambda \log(\max_{1 \leq i \leq r} |X_i|),$$

it holds that

$$\mathrm{KL}\left(p_\theta \,\|\, \hat{q}_\lambda\right) \leq \mathrm{KL}\left(p_\theta \,\|\, q_\lambda\right).$$

Boundedness of $f(T(x))$

Denote $N = 2^n$ and sort all possible points in $\mathcal{B}_n = \{s_1, \ldots, s_N\}$ such that $f(s_1) \leq f(s_2) \leq \cdots \leq f(s_N)$. The bounds of $f(T(x))$ and $\mathbb{E}_{p_\theta}[f(T(x))]$, for a large probability, are not related to samples $s_{M+1}, s_{M+2}, \ldots, s_N$ for an integer M.

Proposition 3

Suppose that the cardinality of each neighborhood $\mathcal{N}(s_i)$ is fixed to be $|\mathcal{N}(s_i)| \geq X \geq n+1$ and all elements in $\mathcal{N}(s_i)$ except s_i are chosen uniformly at random from $\mathcal{B}_n \backslash \{s_i\}$. For $\delta \in (0,1)$, let $M = \left\lceil \frac{\log(N/\delta)}{X-1} N \right\rceil + 1$. Then, with probability at least $1-\delta$ over the choice of $T(x)$, it holds:

1) $f(T(x)) \in [f(s_1), f(s_M)],\ \forall x \in \mathcal{B}_n$;
2) $\mathbb{E}_{p_\theta}[f(T(x))] \leq \sum_{i=1}^{M-1} p_\theta(s_i) f(s_i) + (1 - \sum_{i=1}^{M-1} p_\theta(s_i)) f(s_M) \leq f(s_M)$.

Convergence of MCPG

Assumption: Let $\phi(x;\theta) = \log p_\theta(x|\mathcal{P})$. There exists some constants $M_1, M_2, M_3 > 0$ such that, for any $x \in \mathcal{B}_n$,

1. $\sup_{\theta\in\mathrm{Re}^d} |\phi(x;\theta)| \le M_1$,
2. $\sup_{\theta\in\mathrm{Re}^d} \|\nabla_\theta\phi(x;\theta)\| \le M_2$,
3. $\|\nabla_{\theta_1}\phi(x;\theta) - \nabla_{\theta_2}\phi(x;\theta)\| \le M_3\|\theta_1 - \theta_2\|, \forall\theta_1, \theta_2, \in \mathrm{Re}^d$.

Theorem 3

Let the assumption holds and $\{\theta_t\}$ be generated by MCPG. If the stepsize is chosen as $\eta^t = \frac{c\sqrt{mk}}{\sqrt{t}}$ with $c \le \frac{1}{2l}$, then we have

$$\min_{1\le t\le\tau} \mathbb{E}\left[\|\nabla_\theta L_\lambda(\theta^t)\|^2\right] \le O\left(\frac{\log\tau}{\sqrt{mk\tau}} + \frac{1}{m^2}\right).$$

Parameterization of sampling policy

- Mean field (MF) approximation:

$$p_\theta(x|\mathcal{P}) = \prod_{i=1}^{n} \mu_i^{(1+x_i)/2}(1-\mu_i)^{(1-x_i)/2}, \quad \mu_i = \phi_i(\theta;\mathcal{P})$$

- Parameterization of μ_i:

$$\mu_i = \phi_i(\theta_i) = \frac{1-2\alpha}{1+\exp(-\theta_i)} + \alpha, \quad 1 \le i \le n.$$

  The probability is scaled to the range $(\alpha, 1-\alpha)$, where $0 < \alpha < 0.5$ is given.

- For problems graph structures, combining advanced neural networks such as GNN can also be a good choice.

- We use the results reported by BLS as benchmark. Denoting UB as the results achieved by BLS and obj as the cut size, the gap reported is defined as follows:

$$\mathrm{gap} = \frac{\mathrm{UB} - \mathrm{obj}}{\mathrm{UB}} \times 100\%.$$

Graph	Nodes	Edges	BLS	MCPG	DSDP	RUN-CSP	PI-GNN	EO	EMADM
G14	800	4,694	**3,064**	**3,064**	2,922	2,943	3,026	3047	3045
G15	800	4,661	**3,050**	**3,050**	2,938	2,928	2,990	3028	3034
G22	2,000	19,990	**13,359**	**13,359**	12,960	13,028	13,181	13215	13297
G49	3,000	6,000	**6,000**	**6,000**	**6,000**	**6,000**	5,918	**6000**	**6000**
G50	3,000	6,000	**5,880**	**5,880**	**5,880**	**5,880**	5,820	5878	5870
G55	5,000	12,468	10,294	**10,296**	9,960	10,116	10,138	10107	10208
G70	10,000	9,999	9,541	**9595**	9,456	-	9,421	8513	9557

Table: Computational results on selected Gset instances. The result is sourced from references.

Outline

Mixed integer linear program

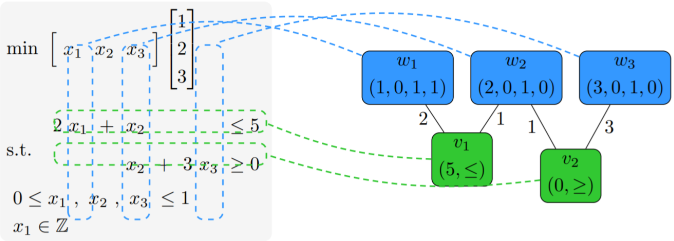
- Mixed-Integer Linear Programs (MILPs) are utilized to solve a myriad of decision-making problems across various practical applications.

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x, \\ \text{s.t.} \quad & Ax \leq b, \\ & l \leq x \leq u, \\ & x \in \mathbb{R}^{n-p} \times \mathbb{Z}^{p}. \end{aligned}$$

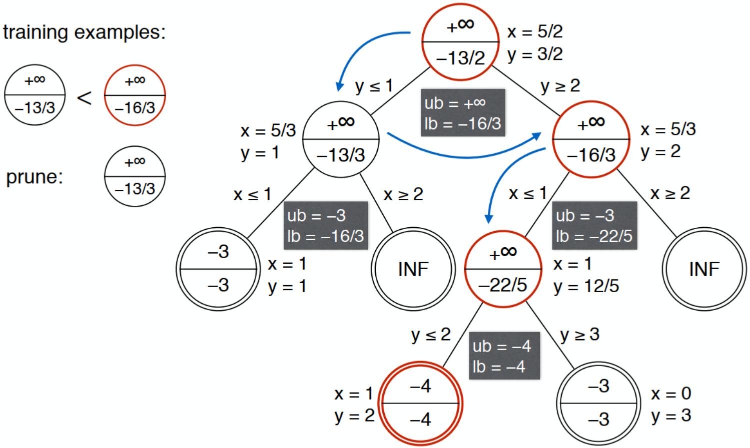
- **Feasibility**: The feasible region is discrete and non-convex, which makes it difficult to analyze and optimization methods hard to design.
- **Complexity**: Even with a relatively small number of variables, the solution space can be exponentially vast due to the integer constraints.
- **Algorithmic strategy**: Preprocessing, Branching Strategies, Bounding Strategies, Cut Generation, Heuristic ...

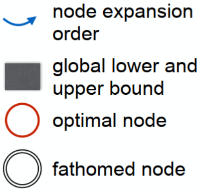
Graph Representation of MILP Instances

- An MILP instance can be represented as a bipartite graph $G = (V \cup W, E)$:
  - **Variable nodes** $w_j \in W$: correspond to variables x_j, each with features:
    - Type of variable (e.g., binary, integer, continuous);
    - Objective coefficient c_j;
    - Bounds $[l_j, u_j]$.
  - **Constraint nodes** $v_i \in V$: represent constraints δ_i, each with features:
    - Constraint type ($\leq$, $=$, or $\geq$);
    - Right-hand side value b_i.
  - **Edges** $(v_i, w_j) \in E$: exist if x_j appears in constraint δ_i, with weight a_{ij}.



Branch and bound





$$
\begin{aligned}
\min\ & -2x - y \\
\text{s.t. } & 3x - 5y \le 0 \\
& 3x + 5y \le 15 \\
& x \ge 0,\ y \ge 0 \\
& x, y \in Z
\end{aligned}
$$

Learning the exact methods

- **Branching Variable Selection:**
  - Branch variable selection determines which fractional variables (also known as candidates) to branch the current node into two child nodes.
  - Nair et al.(2021) encode MIP to the GCN as a bipartite graph and compute an initial feasible solution (Neural Diving), then train a GCN to imitate ADMM-based policy for branching (Neural Branching).
- **Node Selection:**
  - The branch-and-bound algorithm recursively divides the feasible set of a problem into disjoint subsets, organized in a tree structure.
  - He et al.(2014) uses imitation learning to train a node selection and a node pruning policy to speed up the tree search in the B&B process.
- **Cutting Plane:**
  - Cuts serve as the purpose of reducing the LP solution space, which might lead to a smaller tree in the branch-and-cut algorithm.
  - Tang et al. (2020) train a RL agent for sequentially selecting cutting planes.

Outline

Routing problems

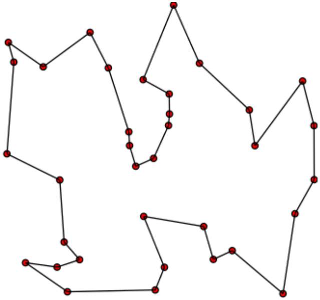
- Travelling Salesman Problem (TSP)
  - Given a fully connected graph with node coordinates $\{x_i\}_{i=1}^n$, the goal is to find a tour that visits each node exactly once and returns to the starting point, while minimizing the total travel distance.
  - Permutaion formulation

$$\min_{\pi} L(\pi) := \sum_{i=1}^{n-1} \|x_{\pi(i+1)} - x_{\pi(i)}\| + \|x_{\pi(1)} - x_{\pi(n)}\|.$$

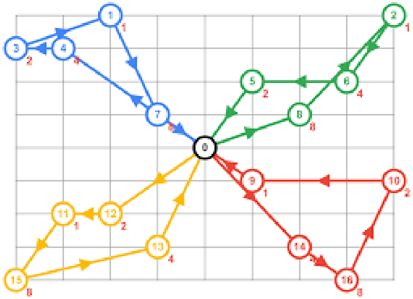
- Capacitated Vehicle Routing Problem (CVRP)
  There are n customers, each with a demand δ_i, to be served by a fleet of identical vehicles with capacity D, all starting and ending at a common depot. The objective is to find the shortest possible set of routes such that every customer is visited exactly once, and the total demand on each route does not exceed the vehicle capacity.

Example tours

- Travelling Salesman Problem (TSP)



- Vehicle Routing Problem (VRP)



- NP-hard combinatorial problem with a wide range of applications!

Overview: machine learning for routing problems

- **Learning to construct**: iteratively add nodes to the partial solution.
  - Pointer Network was first proposed by Vinyals et al. based on Recurrent Neural Networks and supervised learning.
  - The Graph Neural Networks were then leveraged for graph embedding (Dai et al.) and faster encoding (Drori et al.) under reinforcement learning framework.
  - Later, the Attention Model (AM) was proposed by Kool et al.
  - Policy Optimization with Multiple Optima (POMO) significantly improved AM with diverse rollouts and data augmentations (Kwon et al.).
  - Efficient Active Search (EAS) helps to get out of local optima by updating a small subset of pre-trained model parameters on each test instance (Hottung et al.), which could be further boosted if coupled with Simulation Guided Beam Search (SGBS) by Choo et al., achieving better generalization performance.
  - Light Encoder and Heavy Decoder (LEHD) model is proposed by Luo et al. with stronger generalization to large-scale instances sizes.

Overview: machine learning for routing problems

- **Learning to search**: iteratively refine a solution to a new one — a search process.
  - NeuRewriter (Chen et al.) and L2I (Lu et al.) relied heavily on traditional local search algorithms with long run time.
  - Hottung and Tierney proposed the Neural large neighborhood search (NLNS) solver improving upon them by controlling a ruin-and-repair process using a deep model.
  - Several L2S solvers focused on controlling k-opt heuristic within RL training: self-attention-based policy (Wu et al.), Dual-Aspect Collaborative Attention (Ma et al.), Synthesis Attention (Ma et al.) , GNN+RNN-based policy (Costa et al.).
- **Learning to predict**: guide the search by predicting critical information.
  - Joshi et al. proposed using GNN models to predict heatmaps that indicate probabilities of the presence of an edge, which then uses beam search to solve TSP.
  - The GLS solver (Hudson et al.) used GNN to guide the local search heuristics.
  - The DIFUSCO solver (Sun et al.) proposed to replace those GNN models with diffusion models in generating heatmaps.

Comparison

- The L2C solvers can produce high-quality solutions within seconds using greedy rollouts; however, they are shown to get trapped in local optima, even when equipped with post-hoc methods, such as sampling, beam search, etc.
- Although L2S solvers strive to surpass L2C solvers by directly learning to search, they are still inferior to those state-of-the-art L2C solvers even when given prolonged run time.
- Compared to L2C or L2S solvers, L2P solvers exhibit better scalability for large instances; however, L2P solvers are mostly limited to supervised learning and TSP only, due to challenges in preparing training data and the ineffectiveness of heatmaps in handling VRP constraints.

Construct a path

- A solution $\pi = (\pi_1, \ldots, \pi_n)$ is a permutation of the nodes $\{1, \ldots, n\}$.
- Given a problem instance s, the stochastic policy for selecting a solution π is parameterized by θ as

$$p_\theta(\pi|s) = \prod_{t=1}^{n} p_\theta(\pi_t|s, \pi_{1:t-1}).$$

- The encoder produces embeddings of all input nodes, where an instance s is encoded by features x_i on each node i.
- The decoder produces the sequence π of input nodes, one nodes at a time, which takes as input the encoder embeddings and a problem specific mask and context.

Multi-head attention mechanism

- The multi-head attention mechanism starts by linearly projecting input sequences Q, K, V into H distinct subspaces using learned projection matrices W_j^Q, W_j^K, W_j^V:

$$Q_j = QW_j^Q, \quad K_j = KW_j^K, \quad V_j = VW_j^V, \quad j = 1, \ldots, H.$$

- Attention weights are obtained via a scaled dot-product between projected queries and keys, followed by a softmax operation:

$$A_j = \mathrm{Softmax}\left(\frac{Q_j K_j^{\mathrm{T}}}{\sqrt{d_k}} + M\right), \quad j = 1, \ldots, H,$$

  where d_k represents the dimension of the keys and M is an optional attention mask that can be used to prevent attending to certain positions.

Multi-head attention mechanism

- Using these attention weights, the mechanism computes a weighted sum of the projected values, yielding the output of each attention head:

$$Z_j = A_j V_j, \quad j = 1, 2, \ldots, H.$$

- Finally, the outputs from all attention heads are concatenated and linearly projected using a learned output matrix W^O, forming the final multi-head attention output:

$$\mathrm{MHA}(Q, K, V; M) = \mathrm{Concat}(Z_1, \ldots, Z_H) W^O.$$

Encoder

- The encoder computes the initial embeddings $h_i^{(0)} \in \mathbb{R}^{d_h}$ from node features x_i using a linear transformation:

$$h_i^{(0)} = W^{(0)} x_i + b^{(0)}, \quad i = 1, \ldots, n.$$

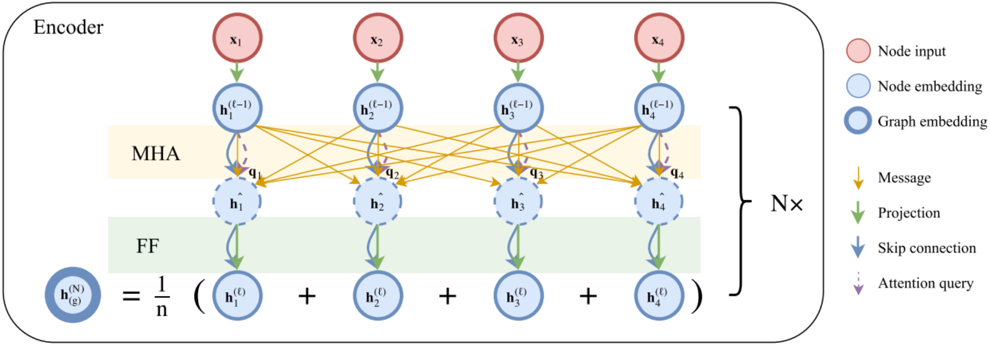
- Stacking these embeddings forms $\boldsymbol{h}^{(0)} \in \mathbb{R}^{n \times d_h}$. The encoder then refines them through L attention layers, each consisting of a multi-head attention (MHA) layer and a node-wise fully connected feed-forward (FF) layer:

$$\hat{\boldsymbol{h}}^{(\ell)} = \mathrm{BN}^{l}\left(\boldsymbol{h}^{(\ell-1)} + \mathrm{MHA}^{(\ell)}\left(\boldsymbol{h}^{(\ell-1)}, \boldsymbol{h}^{(\ell-1)}, \boldsymbol{h}^{(\ell-1)}\right)\right),$$
$$\boldsymbol{h}^{(\ell)} = \mathrm{BN}^{\ell}\left(\hat{\boldsymbol{h}}^{(\ell)} + \mathrm{FF}^{(\ell)}\left(\hat{\boldsymbol{h}}^{(\ell)}\right)\right).$$

- The graph-level representation is the mean of the final node embeddings:

$$\bar{h}^{(L)} = \frac{1}{n} \sum_{i=1}^{n} h_i^{(L)}.$$

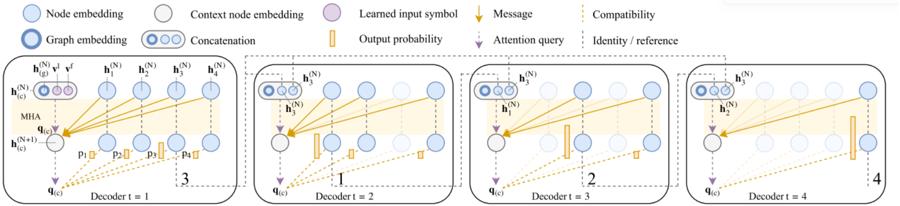
Encoder



Encoder
x_1
x_2
x_3
x_4
$h_1^{(\ell-1)}$
$h_2^{(\ell-1)}$
$h_3^{(\ell-1)}$
$h_4^{(\ell-1)}$
MHA
q_1
q_2
q_3
q_4
$\hat{h}_1$
$\hat{h}_2$
$\hat{h}_3$
$\hat{h}_4$
FF
$h_{(g)}^{(N)} = \frac{1}{n}\left(h_1^{(\ell)} + h_2^{(\ell)} + h_3^{(\ell)} + h_4^{(\ell)}\right)$
N×
Node input
Node embedding
Graph embedding
Message
Projection
Skip connection
Attention query

Decoder

- During decoding, the graph is augmented with a special context node (c) to represent the decoding context.
- The decoder computes an attention (sub)layer on top of the encoder, but with messages only to the context node for efficiency.
- The final probabilities are computed using a single-head attention mechanism.



Context embedding

- The context vector of the decoder at time t consists of the embedding of the graph $\bar{h}^{(L)}$, the previous (last) node π_{t-1} and the first node π_1:

$$h_{(c)} = \begin{cases} [\bar{h}^{(L)}, h^{(L)}_{\pi_{t-1}}, h^{(L)}_{\pi_1}], & t > 1, \\ [\bar{h}^{(L)}, v^1, v^2] & t = 1. \end{cases}$$

- The context embedding $h'_{(c)}$ is computed using a single masked cross-attention layer, where the context vector serves as the query, while the node embeddings provide the keys and values:

$$h'_{(c)} = \mathrm{MHA}(h_{(c)}, \boldsymbol{h}, \boldsymbol{h}; M_t).$$

The mask vector M_t encodes node availability at time t, with $M_t(i) = 0$ for unvisited nodes and $M_t(i) = -\infty$ for visited nodes.

Calculation of probabilities

- The logits are obtained by a single attention head:

$$\boldsymbol{z} = \frac{(\boldsymbol{h}^{(L)} W^K) h'_{(c)}}{\sqrt{d_k}},$$

where the matrix $\boldsymbol{h}^{(L)} W^K$ is precomputed only once as cache during the overall decoding process.

- The conditional probability distribution over available nodes is computed using a softmax:

$$p_\theta(\cdot \mid s, \pi_{1:t-1}) = \text{Softmax}\left(C \cdot \tanh(\boldsymbol{z}) + M_t\right),$$

where the tanh clipping constant $C > 0$ serves in improving the exploration.

Attention model for the CVRP

- **Encoder**: Let $\hat{\delta}_i$ be the normalized demand of the node i.

$$h_i^{(0)} = \begin{cases} W_0^{(0)} x_i + b_0^0, & i = 0, \\ W^{(0)}[x_i, \hat{\delta}_i], & i = 1, \ldots, n. \end{cases}$$

- **Capacity constraints**: Keep track of the remaining demands $\hat{\delta}_{i,t}$ for the nodes $i \in \{1, \ldots, n\}$ and remaining vehicle capacity $\hat{D}_t$ at time t. At $t = 1$, these are initialized as $\hat{\delta}_{i,t} = \hat{\delta}_i$ and $\hat{D}_t = 1$.

$$\hat{\delta}_{i,t+1} = \begin{cases} \max(0, \hat{\delta}_{i,t} - \hat{D}_t), & \pi_t = i, \\ \hat{\delta}_{i,t}, & \pi_t \neq i. \end{cases} \qquad \hat{D}_{t+1} = \begin{cases} \max(0, \hat{D}_t - \hat{\delta}_{\pi_t,t}), & \pi_t \neq 0, \\ 1, & \pi_t = 0. \end{cases}$$

Attention model for the CVRP

- **Decoder context**: The context for the decoder for the VRP at time t is the current/last location π_{t-1} and the remaining capacity $\hat{D}_t$.

$$h_{(c)} = \begin{cases} [\bar{h}^{(L)}, h^{(L)}_{\pi_{t-1}}, \hat{D}_t], & t > 1, \\ [\bar{h}^{(L)}, h^{(L)}_0, \hat{D}_t] & t = 1. \end{cases}$$

- **Masking**: In the decoder layers, the masking rules are defined as follows: for the depot node 0, it is masked (i.e., $M_t(0) = -\infty$) if and only if the current step $t = 1$ or the previous node π_{t-1} is the depot itself. For any customer node $j \neq 0$, it is masked (i.e., $M_t(j) = -\infty$) if it has been visited ($\hat{\delta}_{i,t} = 0$) or its demand exceeds the remaining capacity ($\hat{\delta}_{i,t} > \hat{D}_t$).

Reinforcement learning

- **Loss function**:
$$\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[L(\pi)],$$
where $L(\pi)$ is the tour length for TSP.
- **Policy gradient**:
$$\nabla_\theta \mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[(L(\pi) - b(s))\nabla_\theta \log p_\theta(\pi|s)].$$
- **Rollout baseline**:
$$b(s) = L(\pi^{\mathrm{BL}}),$$
where π^{BL} is a solution from a deterministic greedy rollout of the policy p_θ.
- **Optimizer**: Adam.

Policy Optimization with Multiple Optima

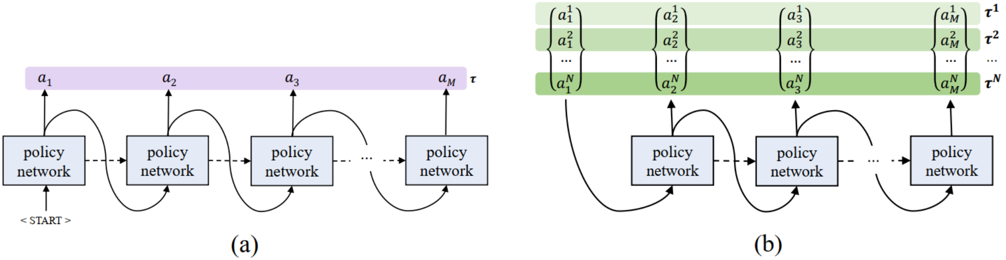
- Symmetry in solving CO problems leads to multiple optima.
- A routing problem contains a loop rather than a sequence, where $(\pi_1, \pi_2, \pi_3, \pi_4)$ is the same as $(\pi_2, \pi_3, \pi_4, \pi_1)$.
- Let a solution trajectory denoted by $\pi = (\pi_1, \ldots, \pi_n)$ and the policy

$$p_\theta(\pi|s) = \prod_{t=1}^{n} p_\theta(\pi_t|s, \pi_{1:t-1}).$$

- In the above equation, the starting nodes π_1 heavily influences the rest of the sequence $(\pi_2, \ldots, \pi_n)$, when in fact any choice for π_1 should be equally good.

Explorations from multiple starting nodes

- Designate N different nodes $\{\pi_1^1, \ldots, \pi_1^N\}$ as starting points for exploration.
- Sample N different solution trajectories $\pi^1, \ldots, \pi^N$ from the policy.
- Apply entropy maximization techniques to improve exploration of the first moves.



Policy gradient with a shared baseline

- A set of solution trajectories $\pi^1, \ldots, \pi^N$ is sampled from the policy $p_\theta(\pi|s)$.
- The policy gradient is approximated by

$$\hat{\nabla}_\theta \mathcal{L}(\theta|s) = \frac{1}{N}\sum_{i=1}^{N}\left(L(\pi^i) - b(s)\right)\nabla_\theta \log p_\theta(\pi^i|s),$$

where $p_\theta(\pi^i|s) = \prod_{t=2}^{n} p_\theta(\pi_t^i|s, \pi_{1:t-1}^i)$.

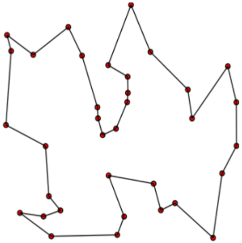
- The shared baseline is taken as the approximation of $\mathbb{E}_{p_\theta(\pi|s)}[L(\pi)]$,

$$b(s) = \frac{1}{N}\sum_{j=1}^{N} L(\pi^j).$$

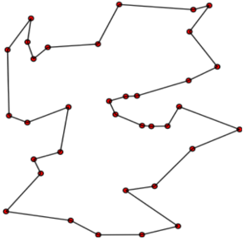
- The shared baseline makes RL training highly resistant to local minima.

Instance augmentation

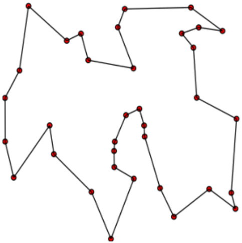
- **Drawback**: N, the number of greedy rollouts one can utilize, cannot be arbitrarily large, as it is limited to a finite number of possible starting nodes.
- **Reformulate the problem**: meet a different problem but arrive at the same solution.
- One can flip or rotate the coordinates of all the nodes in a 2D routing problem and generate another instance, from which more greedy trajectories can be acquired.



0° rotation



90° rotation



180° rotation

Multi-Task vehicle routing problems

- Prevailing neural solvers still need network structures tailored and trained independently for each specific VRP.
- Several VRP variants involve additional practical constraints:
  - **Open route (O):** The vehicle does not need to return to the depot after visiting customers.
  - **Backhaul (B):** We name the customer nodes with $\delta_i > 0$ as linehauls and the ones with $\delta_i < 0$ as backhauls. VRP with backhaul allows the vehicle traverses linehauls and backhauls in a mixed manner, without strict precedence between them.
  - **Duration Limit (L):** To maintain a reasonable workload, the cost (i.e., length) of each route is upper bounded by a predefined threshold.
  - **Time Window (TW):** Each node $v_i \in V$ is associated with a time window $[e_i, l_i]$ and a service time s_i. A vehicle must start serving customer v_i in the time slot from e_i to l_i.

Mixture of Experts

- An MoE layer consists of
  1. m experts $\{E_1, E_2, \ldots, E_m\}$, each of which is a linear layer or FFN with independent trainable parameters.
  2. A gating network G parameterized by W_G, which decides how the inputs are distributed to experts.

$$\mathrm{MoE}(x) = \sum_{j=1}^{m} G(x)_j E_j(x).$$

- A sparse vector $G(x)$ only activates a small subset of experts with partial model parameters, and hence saves the computation.
- A TopK operator can achieve such sparsity by only keeping the K-largest values while setting others as the negative infinity.

$$G(x) = \mathrm{Softmax}(\mathrm{TopK}(x \cdot W_G)).$$

- It jointly optimizes all trainable parameters θ, with the objective formulated as follows

$$\min_{\theta} \mathcal{L} = \mathcal{L}_a + \alpha \mathcal{L}_b.$$

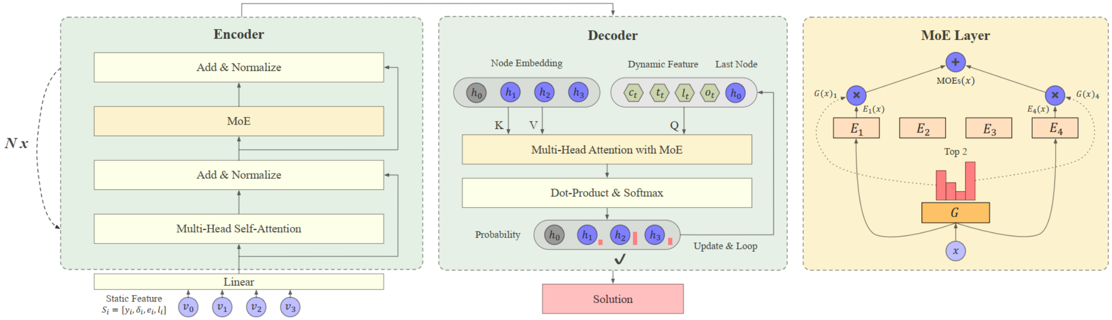
- $\mathcal{L}_a = \mathbb{E}_{\pi \sim p_\theta} [L(\pi)]$ denotes the original loss function of the VRP solver.
- $\mathcal{L}_b$ denotes the the auxiliary loss used to ensure load-balancing in MoEs.

$$I(X) = \sum_{x \in X} G(x),$$

$$D(X)_j = \sum_{x \in X} \Phi \left(\frac{(x \cdot W_G) - \phi(H'_x, k, j)}{\mathrm{Softplus}((x \cdot W_{noise})_j)} \right),$$

$$\mathcal{L}_b = \mathrm{Var}(I(X))^2 + \mathrm{Var}(D(X))^2.$$

MVMoE



- Despite MVMoE presents the first attempt towards a large VRP model, the scale of parameters is still far less than LLMs.

Failure in TSPTW

- The success of the masking mechanism in routing problems relies on
  - the feasibility of the entire solution can be properly decomposed into the feasibility of each node selection step;
  - ground truth masks are easily obtainable for each step.
- However, such assumptions may fail in some routing problems, such as travelling salesman problem with time windows (TSPTW).
- Once a node is selected, the decision becomes irreversible, potentially leading to infeasible situations after several steps.

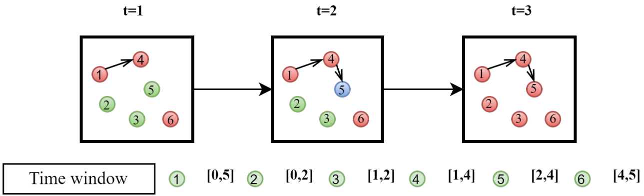
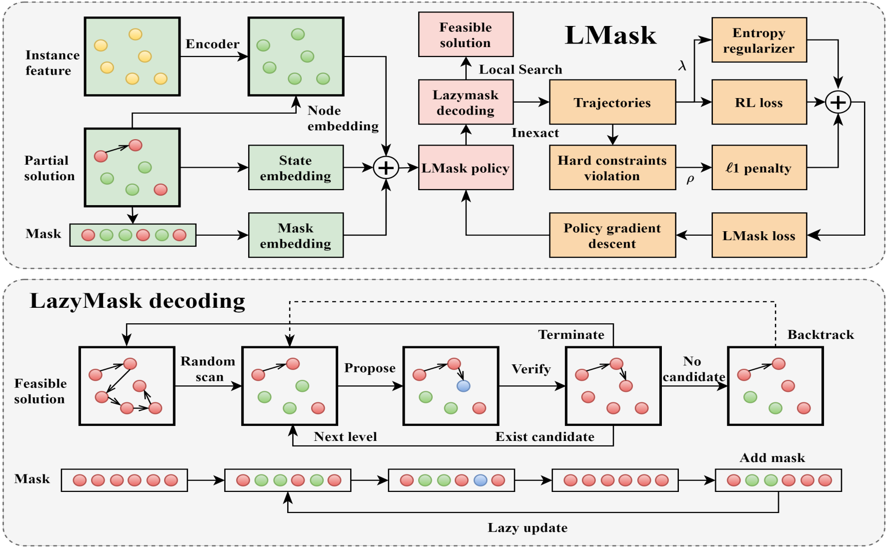


Figure: No node can be selected to satisfy the time windows.

LMask


LMask
Instance feature
Encoder
Node embedding
Partial solution
State embedding
Mask
Mask embedding
Feasible solution
Local Search
Lazymask decoding
Inexact
LMask policy
Trajectories
Hard constraints violation
Policy gradient descent
Entropy regularizer
RL loss
$\ell 1$ penalty
LMask loss
λ
ρ
LazyMask decoding
Feasible solution
Random scan
Propose
Verify
No candidate
Terminate
Backtrack
Next level
Exist candidate
Mask
Add mask
Lazy update

Numerical results

Table 1: Results on synthetic TSPTW datasets.

Nodes		$n = 50$					$n = 100$				
	Method	**Infeasible Sol.**	**Infeasible Inst.**	**Obj.**	**Gap**	**Time**	**Infeasible Sol.**	**Infeasible Inst.**	**Obj.**	**Gap**	**Time**
Easy	PyVRP	-	0.00%	7.31	*	1.7h	-	0.00%	10.19	*	4.3h
	LKH	-	0.00%	7.31	0.00%	1.9h	-	0.00%	10.21	0.29%	7.2h
	PIP	0.28%	0.01%	7.51	2.70%	9s	0.16%	0.00%	**10.57**	**3.57%**	29s
	PIP-D	0.28%	**0.00%**	7.50	2.57%	10s	**0.05%**	0.00%	10.66	4.41%	31s
	LMask	**0.09%**	0.01%	**7.49**	**2.55%**	8s	0.08%	0.00%	10.62	4.23%	14s
Medium	PyVRP	-	0.00%	13.03	*	1.7h	-	0.00%	18.72	*	4.3h
	LKH	-	0.00%	13.02	0.00%	2.9h	-	0.01%	18.74	0.16%	10.3h
	PIP	4.82%	1.07%	13.41	3.07%	10s	4.35%	0.39%	19.62	4.73%	29s
	PIP-D	4.14%	0.90%	13.46	3.45%	9s	3.46%	0.03%	19.80	5.70%	31s
	LMask	**0.33%**	**0.03%**	**13.36**	**2.53%**	9s	**0.49%**	**0.00%**	**19.57**	**4.52%**	15s
Hard	PyVRP	-	0.00%	25.61	*	1.7h	-	0.01%	51.27	*	4.3h
	LKH	-	0.52%	25.61	0.00%	2.3h	-	0.95%	51.27	0.00%	1d8h
	PIP	5.65%	2.85%	25.73	1.12%	9s	31.74%	16.68%	51.48	0.80%	28s
	PIP-D	6.44%	3.03%	25.75	1.20%	9s	13.60%	6.60%	51.43	0.68%	31s
	LMask	**2.40%**	**1.28%**	**25.70**	**0.08%**	10s	**5.63%**	**2.31%**	**51.34**	**0.14%**	31s