Planning and Optimal Control

Learning to Optimize -
RL for Discrete Optimization

Jonas Falkner, Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Syllabus

		A. Models for Sequential Data
Tue. 22.10.	(1)	1. Markov Models
Tue. 29.10.	(2)	2. Hidden Markov Models
Tue. 5.11.	(3)	3. State Space Models
Tue. 12.11.	(4)	3b. (ctd.)
		B. Models for Sequential Decisions
Tue. 19.11.	(5)	1. Markov Decision Processes
Tue. 26.11.	(6)	1b. (ctd.)
Tue. 3.12.	(7)	1c. (ctd.)
Tue. 10.12.	(8)	2. Monte Carlo and Temporal Difference Methods
Tue. 17.12.	(9)	3. Q Learning
Tue. 24.12.	—	— *Christmas Break* —
Tue. 7.1.	(10)	4. Policy Gradient Methods
Tue. 14.1.	(11)	5. Cooperative Reinforcement Learning
Tue. 21.1.	(12)	6. Learning to Optimize
Tue. 28.1.	(13)	7. Reinforcement Learning for Games
Tue. 4.2.	(14)	Q&A

Outline

1. Discrete Optimization
2. MDP Formulation
3. Approaches for Direct Construction
4. Approaches for Consecutive Improvement

Outline

1. Discrete Optimization

2. MDP Formulation

3. Approaches for Direct Construction

4. Approaches for Consecutive Improvement

What is Discrete Optimization?

- Compared to continuous optimization, some or all relevant variables are restricted to be discrete variables $x \in \mathcal{D}$, where $\mathcal{D}$ is a set of discrete values.
- Mostly involved with solving Combinatorial Optimization Problems (COP) with the help of e.g. Integer Programming or Constraint Programming.

Common Combinatorial Optimization Problems

- There are a lot of different COPs that are relevant in research and industry.

Common Combinatorial Optimization Problems

- There are a lot of different COPs that are relevant in research and industry.
- Some examples are:
  - Convex Hull
  - Knapsack
  - Graph Problems (Max-Cut, Minimum Vertex Cover, etc.)
  - Routing Problems (TSP, VRP, etc.)
  - many more ...

The Knapsack Problem

- Given a set of items with different attributes (e.g. weight and value) find the subset that maximizes an objective involving one or more attributes substitute to some constraints w.r.t. other attributes.
- A simple example is maximizing the value with a constraint on the total weight of the selected items.

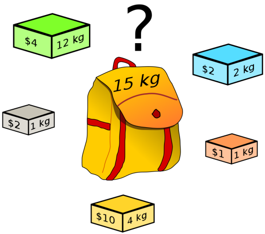
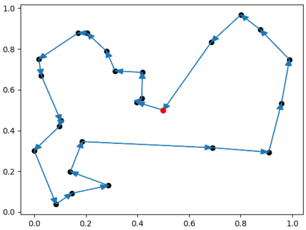


Image source: CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=985491

The Traveling Salesman Problem (TSP)

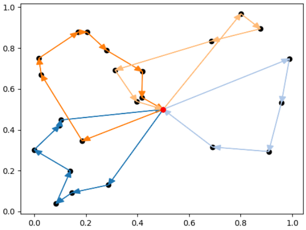
- Given a set N of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin?

- First formulated in 1930, it is one of the most intensively studied problems in optimization and therefore used as a benchmark for many optimization methods.



Vehicle Routing Problems (VRP)

- Generalization of the TSP for more than one vehicle.
- In standard formulation 1 depot and all vehicles have the same capacity.
- Lots of extensions of the VRP to describe more complex problem settings, e.g.
  - Time Windows (TW),
  - Pickup and Delivery (PD),
  - Heterogeneous Fleet,
  - etc.



Challenges

- In general COPs are NP-hard.
  - Cannot be solved in polynomial time but only verified.
- Especially real world routing problems come with many different hard and soft constraints.
  - Just finding a **feasible** solution is already very hard.

Outline

1. Discrete Optimization

2. MDP Formulation

3. Approaches for Direct Construction

4. Approaches for Consecutive Improvement

How can we solve COPs with ML/RL?

How can we solve COPs with ML/RL?

- Learn solution strategies from data/experience!
- Formulate the solution of a COP as MDP.
- For most COPs there exist to ways of formulating them as MDP:
  1. Direct sequential construction of a solution one item at a time,
  2. Consecutive improvement of an existing feasible solution.

Direct sequential construction

The first option we have is to solve the COP by directly constructing a high quality solution:

- We formulate the solution of the COP as a sequence of actions $a_t = n \in N$ where $N \equiv \mathcal{D}$ is the set of the discrete decision variable, which we will denote as set of actions $\mathbf{A}$,
- States $s_t \in \mathbf{S}$ which we assume to sufficiently describe the state of the problem and the current solution at time step t,
- The transition matrix $\mathbf{P}$ in general is deterministic for most problems,
- The reward $\mathbf{r}$ can be given as the additional cost per decision or as quality metric on the final solution,
- Leading to the MDP formulation as $(\mathbf{S}, \mathbf{A}, \mathbf{T}, \mathbf{P}, \mathbf{r})_1$.

Improvement of an existing solution

The second option assumes that we have a first feasible but not optimal solution to the COP and is concerned with finding a better solution:

- We formulate the consecutive improvement of a COP solution as a sequence of actions $a_t \in \mathbf{A}$ where $\mathbf{A}$ is a set of problem dependent improvement operators (heuristics),
- States $s_t \in \mathbf{S}$ which we assume to sufficiently describe the state of the problem and the current solution at time step t,
- The transition matrix $\mathbf{P}$ in general is deterministic for most problems,
- The reward $\mathbf{r}$ is given as the improvement over the current best solution,
- Leading to the MDP formulation as $(\mathbf{S}, \mathbf{A}, \mathbf{T}, \mathbf{P}, \mathbf{r})_2$.

Outline

1. Discrete Optimization

2. MDP Formulation

3. Approaches for Direct Construction

4. Approaches for Consecutive Improvement

How can we embed the problem state?

- Finding a good representation and embedding for the states is of high importance, since we assume that states
  "...sufficiently describe the state of the problem and the current solution..."
- Example of problem components in a TSP:
  - $n = |N|$ cities each with two coordinates (x, y),
  - current partial solution $\pi_t = \{a_1, a_2, ..., a_t\}$,
  - origin city n_0.

Pointer Networks

- One of the first attempts to **directly** solve COPs by Vinyals et al. 2015,
- Employs a sequence-to-sequence encoder-decoder RNN model,
- Uses special **pointer attention** masks,
- Original model is trained in a **supervised fashion**, however later extended to RL training by Bello et al. 2016.

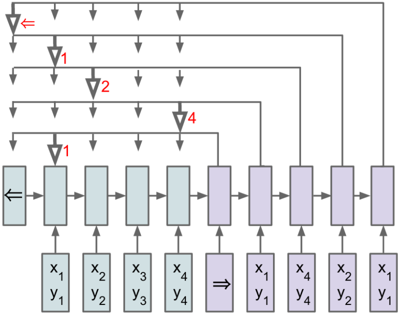


Image source: Vinyals et al. 2015

Pointer Networks (ctd.)

Given an input sequence X, the Pointer Network computes the conditional probability $p(\pi \mid X;\ \theta)$ via the probability chain rule:

$$p(\pi \mid X;\ \theta) = \prod_{t=1}^{n} p(\pi_t \mid X,\ \pi_{1:t-1};\ \theta) \tag{1}$$

where $X = \{x_1, ..., x_n\}$ is a sequence of n vectors (e.g. cities in the TSP) and $\pi = \{\pi_1, ..., \pi_T\}$ is a sequence of indices indexing the elements of X.

The supervised objective is to maximize the conditional probability on a provided training set:

$$\theta^* = \arg\max_{\theta} \sum_{X,\ \pi^{(train)}} \log p(\pi^{(train)} \mid X;\ \theta) \tag{2}$$

Pointer Attention

Let $(e_1, ..., e_n)$ be the encoder and $(d_1, ..., d_t)$ the decoder hidden states. Then the standard attention mask at time step t is computed according to:

$$u_i^{(t)} = \omega^T \tanh(W_1 e_i + W_2 d_t) \qquad i \in (1, ..., n) \tag{3}$$

$$a_i^{(t)} = softmax(u_i^{(t)}) \qquad i \in (1, ..., n) \tag{4}$$

$$d'_t = \sum_{i=1}^{n} a_i^{(t)} e_i \tag{5}$$

For the pointer attention the mask is instead defined as:

$$p(\pi_t \mid X,\ \pi_{1:t-1}) = softmax(u^{(t)}) \tag{6}$$

where the softmax normalizes the vector $u^{(t)}$ (of length n) to be an output distribution over the set of inputs.

Note: ω (vector), W_1 and W_2 (matrices) are learnable parameters of the model.

Why would we like to use RL instead of purely supervised training?

Why would we like to use RL instead of purely supervised training?

- The **performance** of the model is tied to the **quality of the supervised labels**,
- Getting high-quality labeled data is **expensive** and may be **infeasible** for new problem statements,
- We care more about finding a **competitive solution** than **replicating** the results of another algorithm.

Learning Pointer Networks via Policy Gradient

- Extension proposed by Bello et al. 2016
- Employ an Advantage Actor Critic algorithm (A3C, Mnih et al. 2016)

We define the new objective as the expected quality of the solution, given input X:

$$J(\theta \mid X) = \mathbb{E}_{\pi \sim p(\,.\,|X;\,\theta)}\left[L(\pi \mid X)\right] \tag{7}$$

Then the policy gradient is defined as (REINFORCE):

$$\nabla_\theta J(\theta \mid X) = \mathbb{E}_{\pi \sim p(\,.\,|X;\,\theta)}\left[(L(\pi \mid X) - b(X))\,\nabla_\theta \log p(\pi \mid X;\,\theta)\right] \tag{8}$$

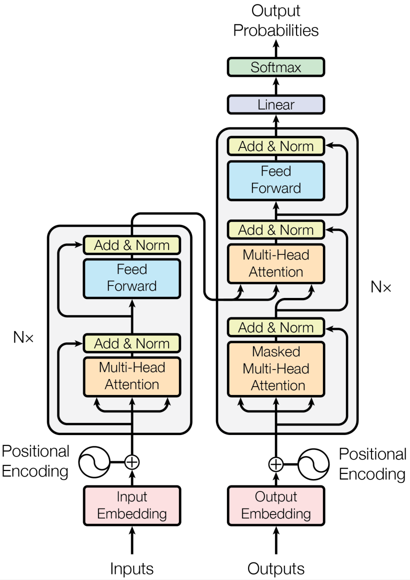
Transformer Networks

- First proposed by Vaswani et al. 2017 for machine translation,
- Adapted by Kool et al. 2019 to solve routing problems.

- Similar to the RL Pointer Network defines a stochastic policy $p(\pi \mid X;\ \theta)$ based on a **Self-Attention** Encoder-Decoder model (transformer),
- and computes a solution π in the same way as in Eq.1:

$$p(\pi \mid X;\ \theta) = \prod_{t=1}^{n} p(\pi_t \mid X,\ \pi_{1:t-1};\ \theta)$$

The transformer architecture of Vaswani et al. 2017.



Self-Attention Encoder (1/5)

Simple **Scaled Dot-Product Attention** is defined as:

$$u_{ij} = \frac{(Wx_i)^T Wx_j}{\sqrt{d_{emb}}} \tag{9}$$

where the input is node features x_i and x_j.

Self-Attention first embeds the node features via a linear projection:

$$\tilde{x}_i = W_{init} x_i \tag{10}$$

Then produces a **query**, **key** and **value** embedding by additional projections:

$$q_i = W_Q \tilde{x}_i, \qquad \kappa_i = W_\kappa \tilde{x}_i, \qquad v_i = W_V \tilde{x}_i. \tag{11}$$

Self-Attention Encoder (2/5)

Then the respective utilities are calculated according to:

$$u_{ij} = \begin{cases} \frac{q_i^T \kappa_j}{\sqrt{d_\kappa}} & \text{if } j \in \mathcal{H}_i \\ -\infty & \text{else} \end{cases} \tag{12}$$

where $-\infty$ prevents message passing between non-adjacent nodes which do not lie in the neighborhood $\mathcal{H}_i$ of node i.

In the next step the attention weights a_{ij} are computed with a softmax:

$$a_{ij} = softmax_j(u_{ij}) = \frac{\exp(u_{ij})}{\sum_{q \in \mathcal{H}_i} \exp(u_{iq})} \tag{13}$$

Self-Attention Encoder (3/5)

Finally there are two ways to compute the new feature representation x_i':

1. **Single-head attention:**

$$x_i' = \sum_j a_{ij} v_j \tag{14}$$

2. **Multi-head attention (MHA):**
   calculates Eq.14 M times with different parameters and reduced dimensions $d_\kappa = d_v = \frac{d_{init}}{M}$. The resulting M vectors $x_i'^{(m)}$, $m \in 1, ..., M$ are then concatenated and projected back:

$$MHA_i = [x_i'^{(1)} : ... : x_i'^{(m)}] W_M \tag{15}$$

Self-Attention Encoder (4/5)

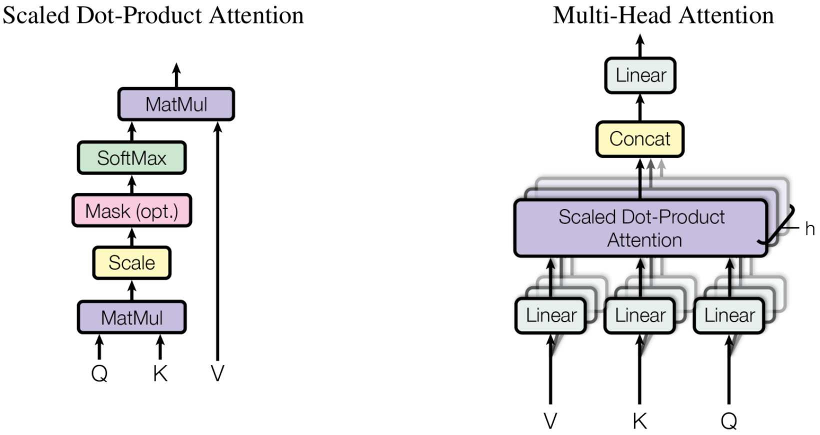


Figure: The single and multi-head attention layers of Vaswani et al. 2017.

Self-Attention Encoder (5/5)

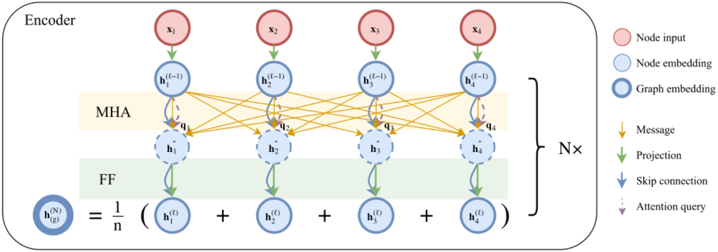


Figure: The Encoder forward of Kool et al. 2019.

Qualitative Advantages of the Self-Attention Encoder

Compared to an RNN Encoder as used in the Pointer Net the Self-Attention Encoder...

- is able to model **dependencies** between nodes or specific node features,
- embedding can be **pre-computed** also for complex problems and the encoder doesn't need to be run for each time step t,
- enforces no order on the input but is **permutation equivariant** (node-embedding) or **invariant** (graph-embedding).

Self-Attention Decoder (1/2)

- The decoder has a similar structure as the encoder. However the query is computed over the **context vector** $h_{(c)}$.
- The context is the concatenation of the graph embedding, the last node in the current tour and the origin (TSP) or capacity (VRP).
- We again do a linear projection $q_{(c)} = W_C h_{(c)}$ and calculate the utilities (compatibility) according to:

$$u_{(c)j} = \begin{cases} \frac{q_{(c)}^T \kappa_j}{\sqrt{d_\kappa}} & \text{if } \mathit{is\_feasible}(j, k) \\ -\infty & \text{else} \end{cases} \tag{16}$$

  by which we can enforce the hard problem constraints at time step t.
- Followed by softmax normalization and MHA as in Eq. 13 and 15.

Self-Attention Decoder (2/2)

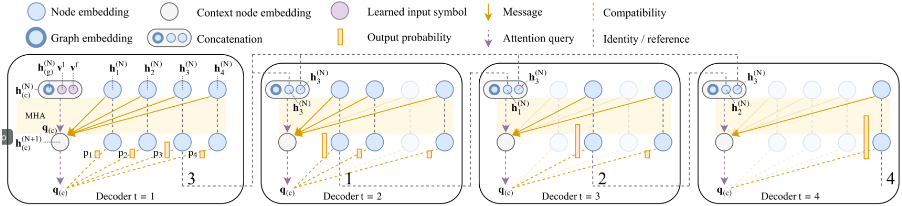


Figure: The Decoder forward of Kool et al. 2019.

Model Comparison

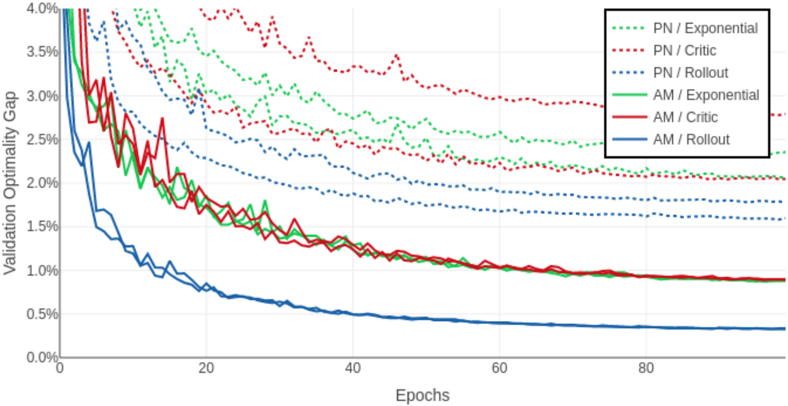


Figure: Comparison of Pointer Net (PN) and Self-Attention Model (AM) on TSP with different policy gradient baselines in Kool et al. 2019. They compare exponential moving average, a learned critic and the cost of a solution from a deterministic greedy rollout of the best policy so far.

Inference Strategies

The optimality of solutions produced by direct construction is not guaranteed. Therefore several advanced inference strategies can be used:

- **Sampling**:
  Simply sample multiple candidate solutions and select the best one,
- **Active Search**:
  Refine the parameters of the stochastic policy p_θ during inference to minimize $\mathbb{E}_{\pi \sim p(\,.\,|X;\,\theta)} [L(\pi \mid X)]$ on a single test input X,
- **Heuristic Post-Optimization**:
  Apply additional improvement heuristics to the solution (e.g. 2-opt),
- **Monte-Carlo Tree Search (MCTS)**:
  Apply MCTS together with heuristic transformations
  (more on MCTS in next lecture).

Outline

1. Discrete Optimization

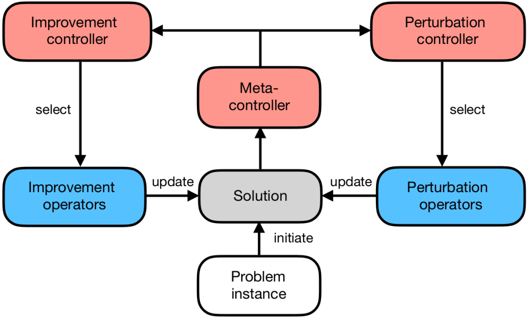
2. MDP Formulation

3. Approaches for Direct Construction

4. Approaches for Consecutive Improvement

Iterative Improvement with RL Controller

- Method proposed by Lu et al. 2019,
- Implements an iterative method that selects an improvement operator (heuristic) and applies it to the existing solution,
- If there is no improvement achieved for several iterations, the solution is perturbed ($\rightarrow$ rule based controller!).



Implementation Details

- Uses a Self-Attention Encoder (like the transformer encoder) to embed the problem state,
- The Decoder is represented by a 2-layer FC network,
- The input to the model at each time step consists of the embedding of the problem and the current solution (tour plan) and a running history of past actions and their effects,
- While the policy network controlling the improvement operators is learned, the perturbations are done in a random fashion.

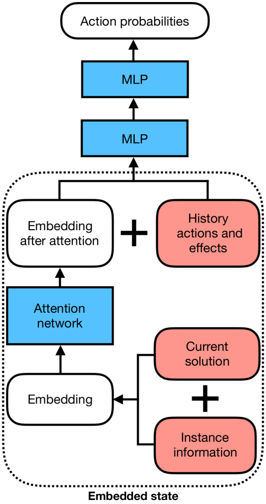


Image: Policy network of Lu et al. 2019

Model Comparison

Method	N=20		N=50		N=100	
	Obj.	Time	Obj.	Time	Obj.	Time
Google OR Tools	6.43	-	11.31	-	17.16	-
AM (greedy)	6.40	1s	10.98	3s	16.80	8s
AM (sampling)	6.25	6m	10.62	28m	16.23	2h
Local Rewrite	6.16	-	10.51	-	16.10	-
LKH	6.14	2h	10.38	7h	15.65	13h
Iterative Improvement	**6.12**	12m	**10.35**	17m	**15.57**	24m

Table: Experiment results on CVRP from *Table: 1* in Lu et al. 2019

Where:
AM: [Kool et al. 2019],
Local Rewrite: [Chen and Tian 2019],
LKH: [Helsgaun]

References

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. *arXiv:1611.09940 [cs, stat]*, November 2016. URL `http://arxiv.org/abs/1611.09940`. arXiv: 1611.09940.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, pages 6278–6289, 2019.

Keld Helsgaun. An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems. page 60.

Wouter Kool, Herke van Hoof, and Max Welling. ATTENTION, LEARN TO SOLVE ROUTING PROBLEMS! page 25, 2019.

Hao Lu, Xingwen Zhang, and Shuang Yang. A Learning-based Iterative Method for Solving Vehicle Routing Problems. September 2019. URL `https://openreview.net/forum?id=BJe1334YDH`.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5866-pointer-networks.pdf`.