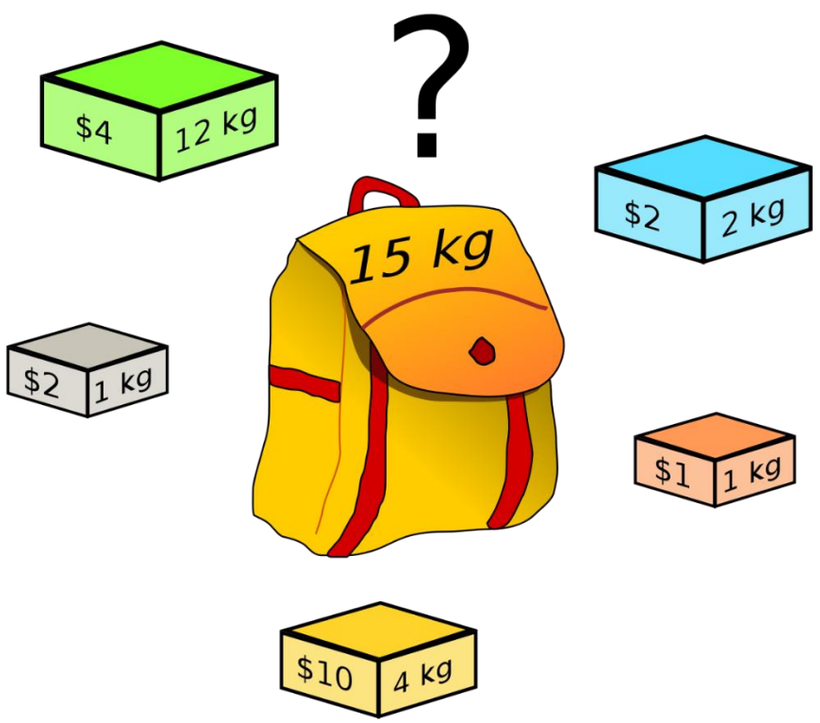
Solving Combinatorial Optimization Problems with Reinforcement Learning

AI Projects #3

Furkan Gürsoy
Batuhan Koyuncu

Combinatorial Optimization

Combinatorial optimization (CO) deals with finding optimal or near-optimal solutions from a set of all possible solutions that is generally too large for an exhaustive search and is defined by certain restrictions.



Knapsack problem is a popular CO problem.

Given that

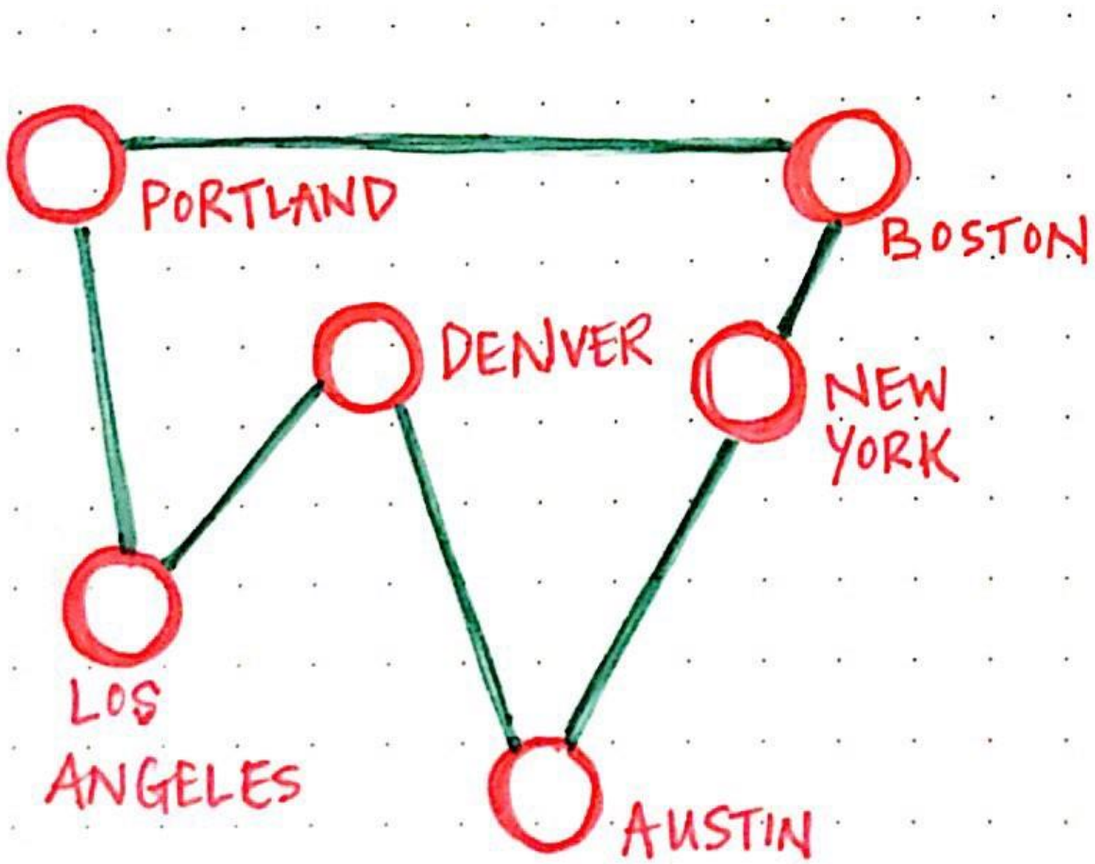
- Each item has value $v_i > 0$ and weight $w_i > 0$.
- The bag can only carry a certain weight.

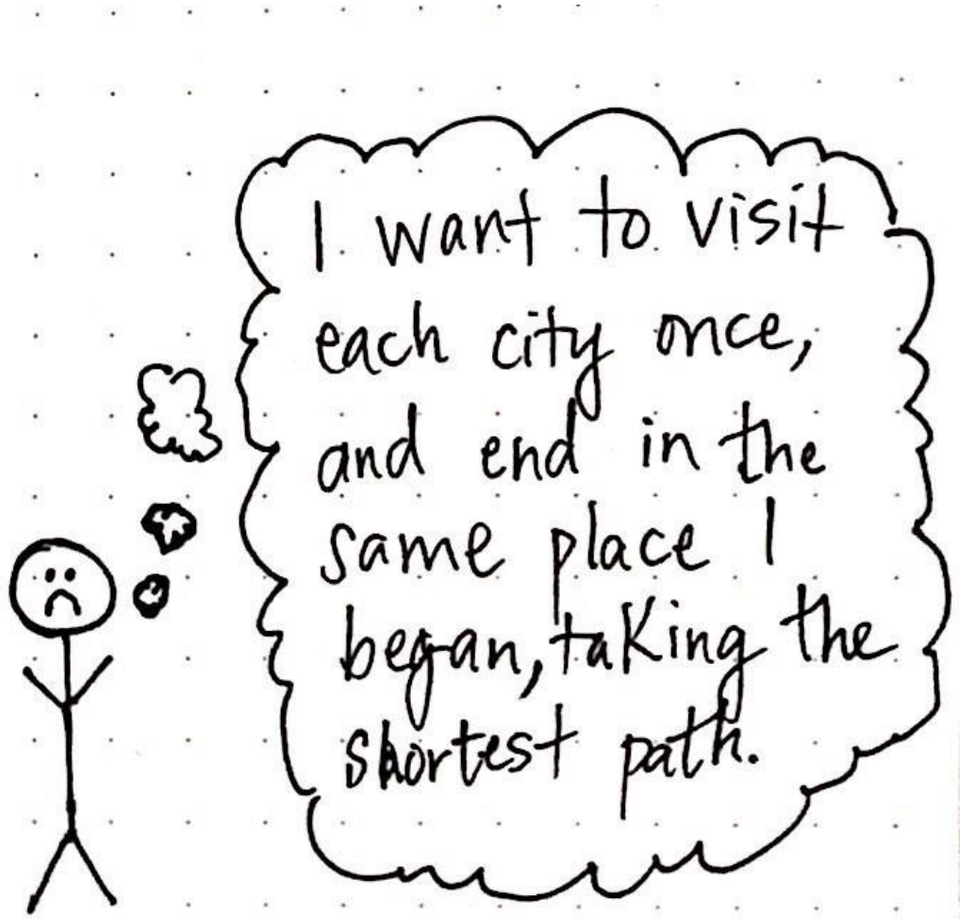
Which items to choose so that the total value is maximized?

Traveling Salesman Problem (TSP)

A TSP is usually concerned with finding the shortest route which passes through every specified point in the problem.

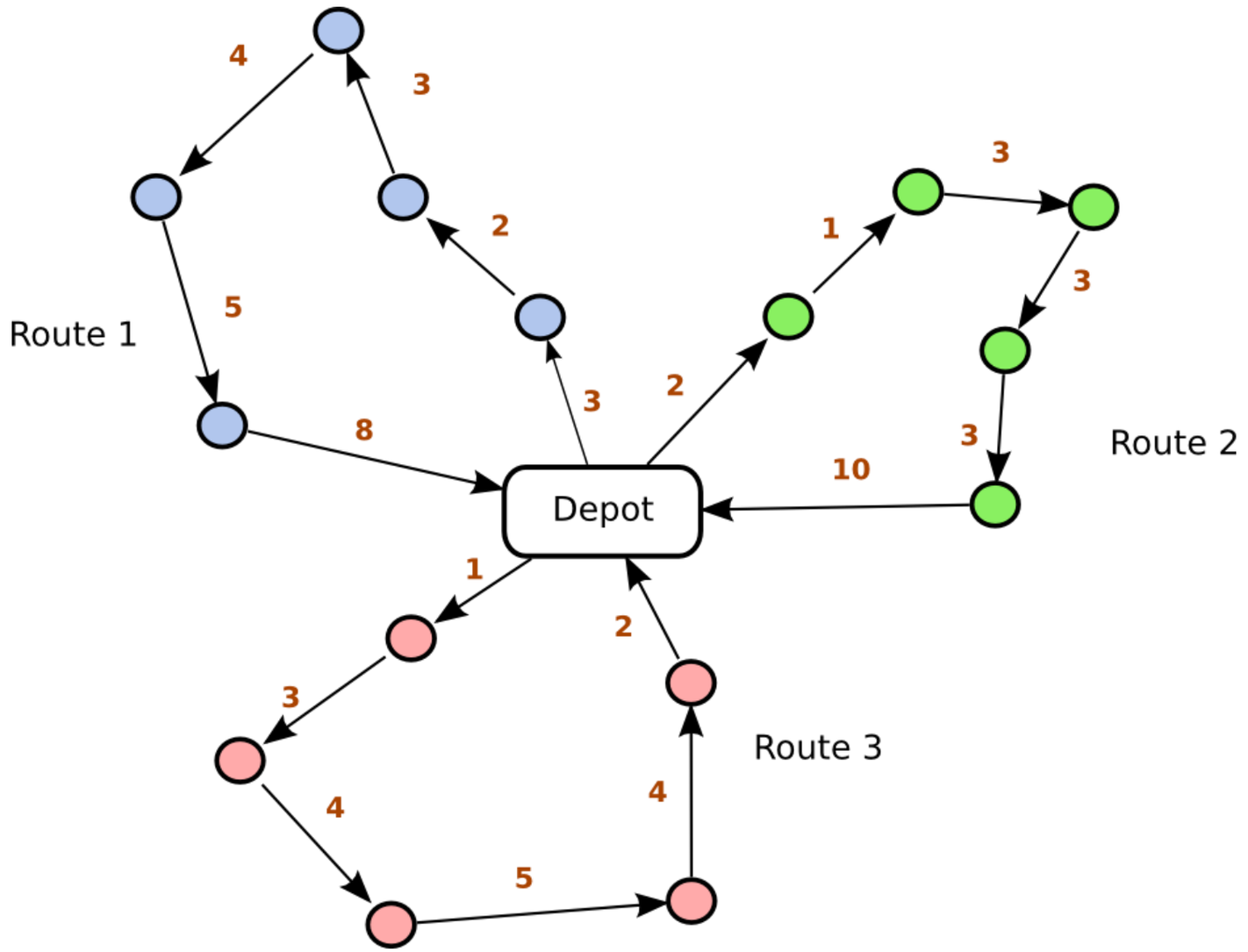
For this study, we focus on Euclidean TSP.





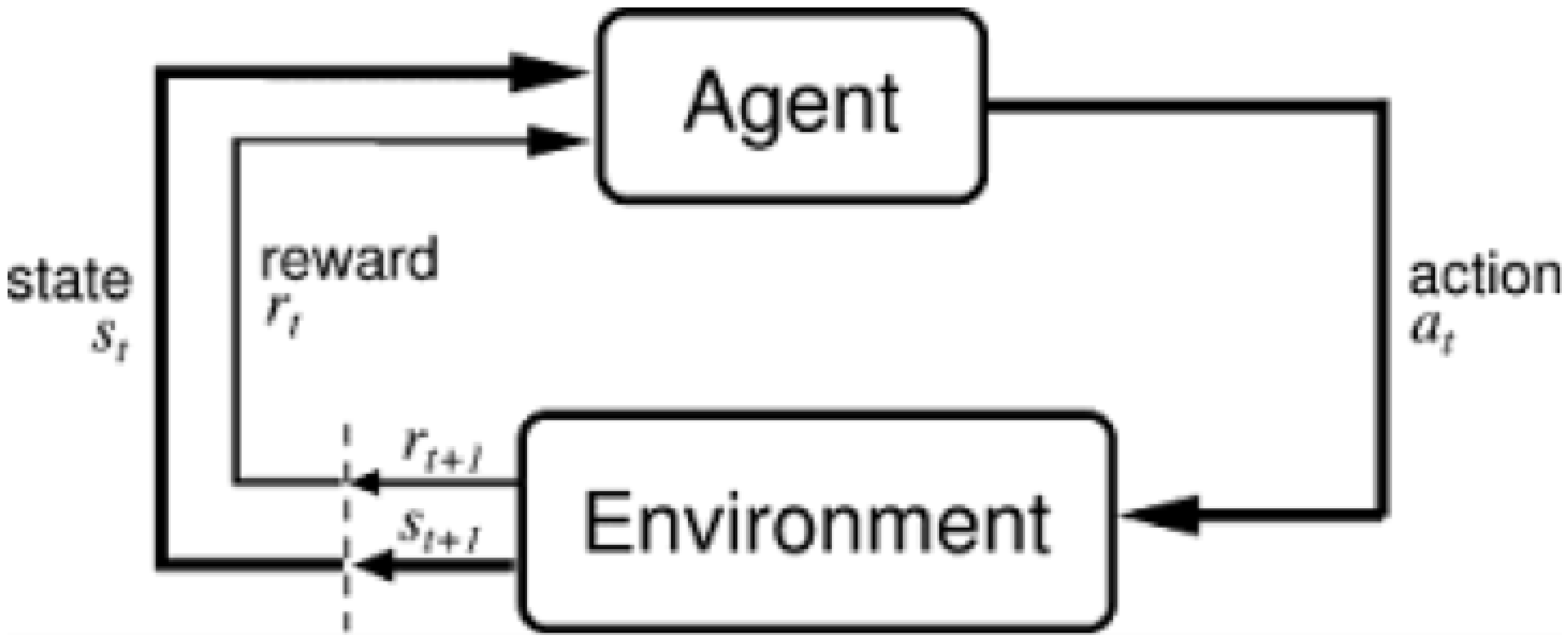
Vehicle Routing Problem

Generalization of TSP, aim is to find the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers.



Reinforcement Learning

RL refers to the goal-oriented algorithms which learn how to maximize an objective function by interacting with the environment and evaluating the resulting rewards.



The Motivation

Combinatorial optimization problems are often solved with hand-crafted algorithms: heuristics or exact methods. These are often costly to develop and difficult generalize!

Can machines be trained to learn such heuristics?

Recently, reinforcement learning-based techniques are proposed to solve various combinatorial optimization problems.

In this work, we focus on solving Euclidean TSP using RL.

A chronology of CO X RL

- **Vinyals et al. (2015).** Pointer network trained by supervised learning using example solutions.

- **Bello et al. (2016).** Pointer network trained by reinforcement learning using Actor-Critic algorithm.

- **Kool et al. (2019).** Use attention network instead of pointer network. Train by reinforcement learning using policy gradient.

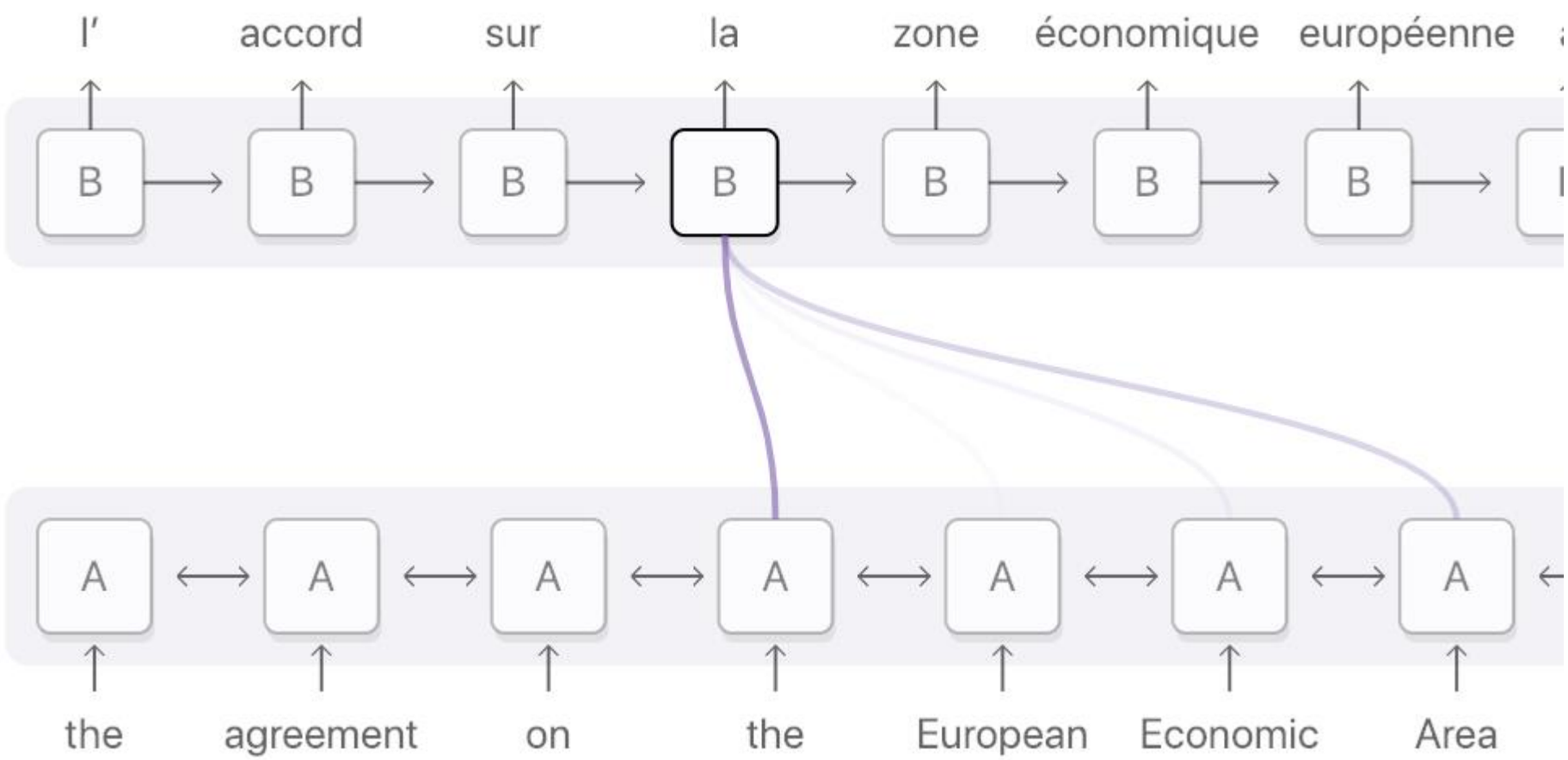
O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. arXiv preprint arXiv:1511.06391, 2015.

Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940, 2016

Kool, W., van Hoof, H., & Welling, M. Attention, Learn to Solve Routing Problems!. International Conference on Learning Representations, 2019.

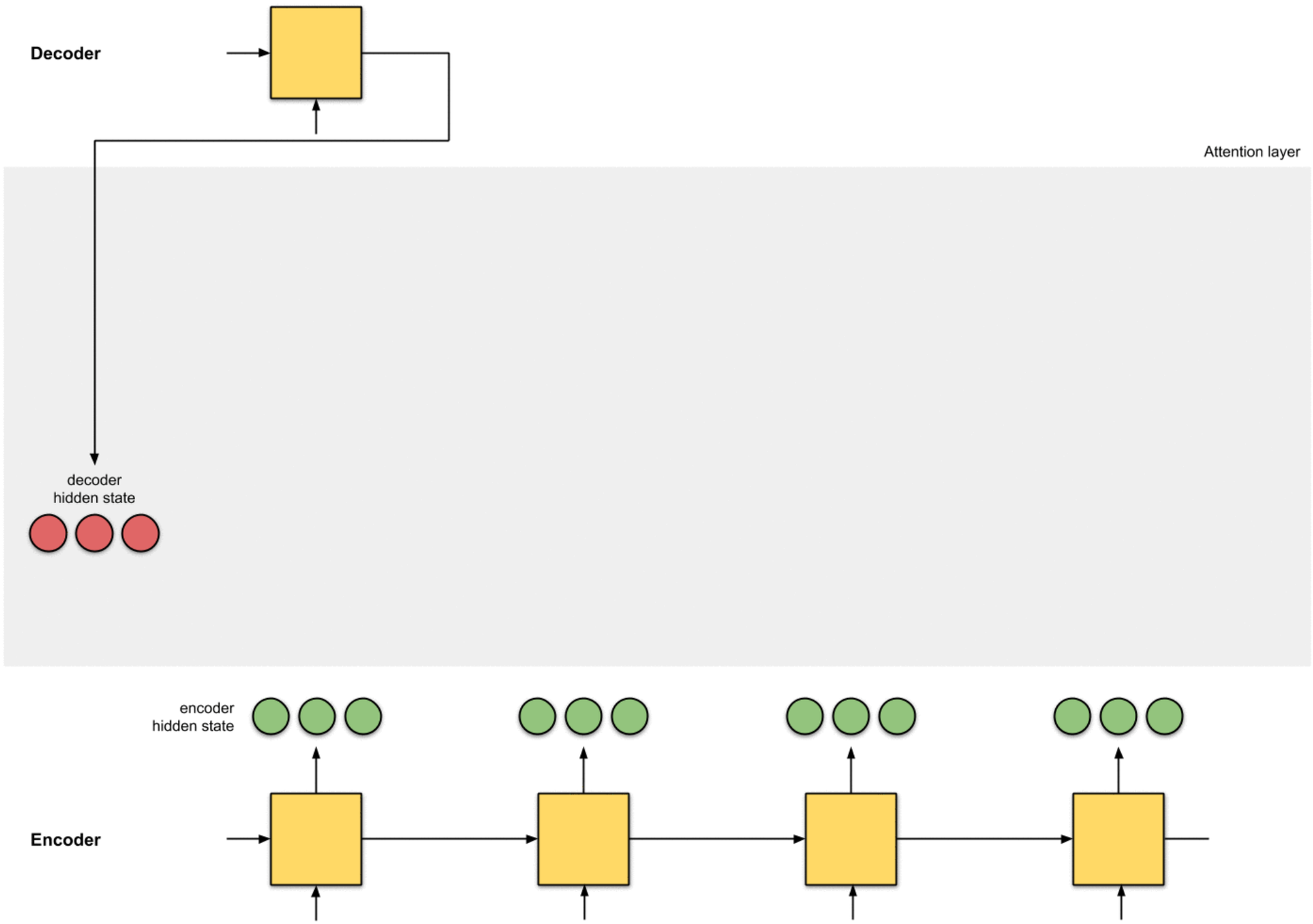
Attention

Attention is an interface between the encoder and decoder that provides the decoder with information from every encoder hidden state.



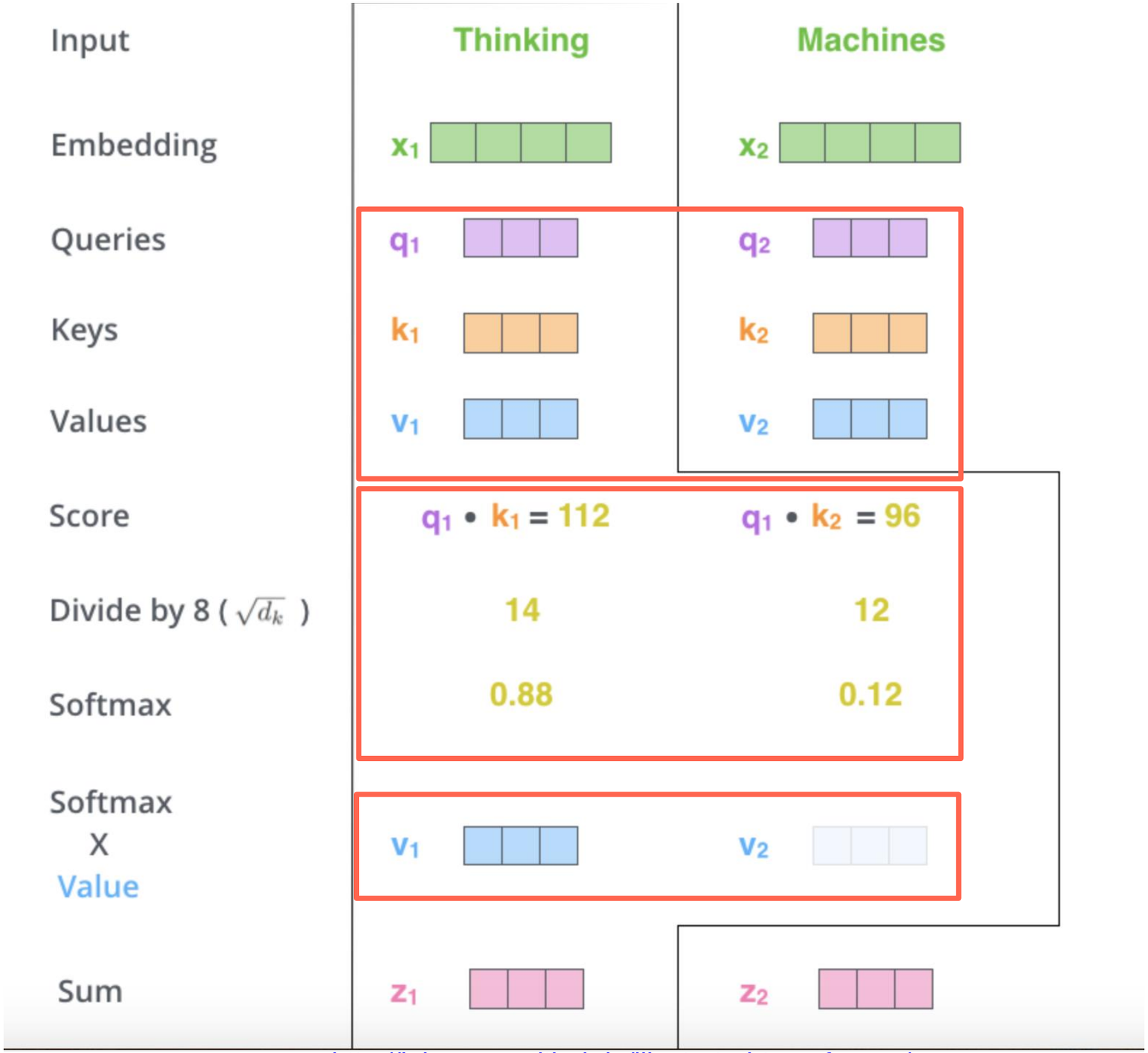
source: https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3

Attention in seq2seq networks



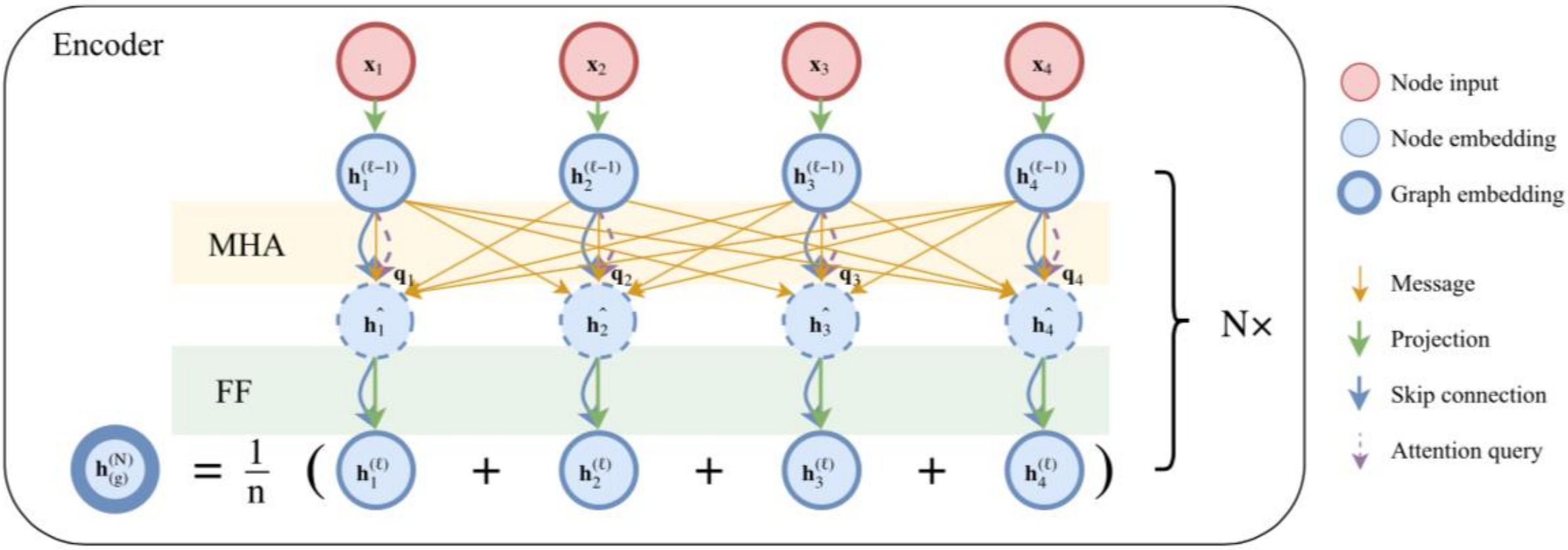
source: https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3#0458

Multihead Attention



source: http://jalammar.github.io/illustrated-transformer/

Model Encoder



source: Wouter Kool, Herke van Hoof, Max Welling,
‘Attention, Learn to Solve Routing Problems!’, ICLR 2019.

Model Decoder

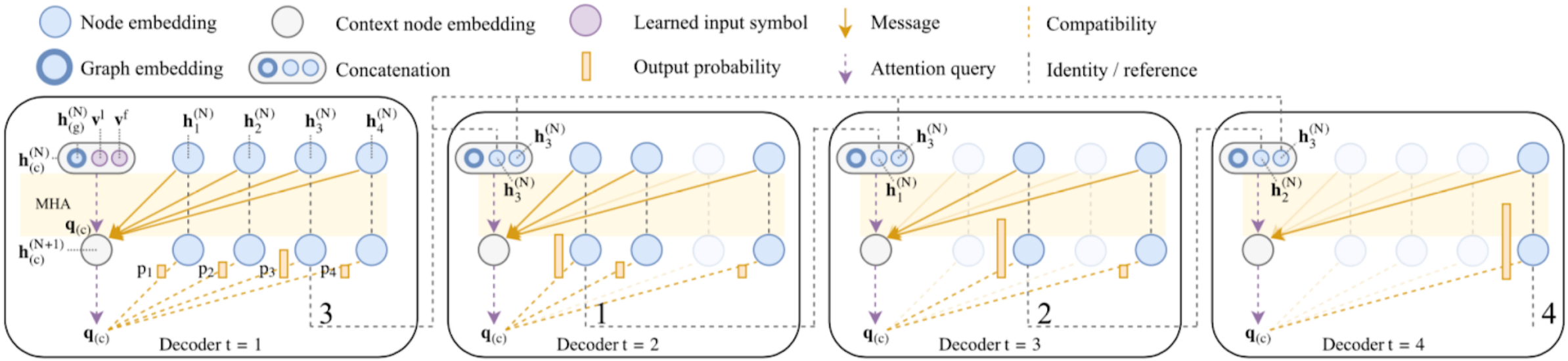


Figure 2: Attention based decoder for the TSP problem. The decoder takes as input the graph embedding and node embeddings. At each time step t, the context consist of the graph embedding and the embeddings of the first and last (previously output) node of the partial tour, where learned placeholders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

source: Wouter Kool, Herke van Hoof, Max Welling,
'Attention, Learn to Solve Routing Problems!', ICLR 2019.

Model Decoder

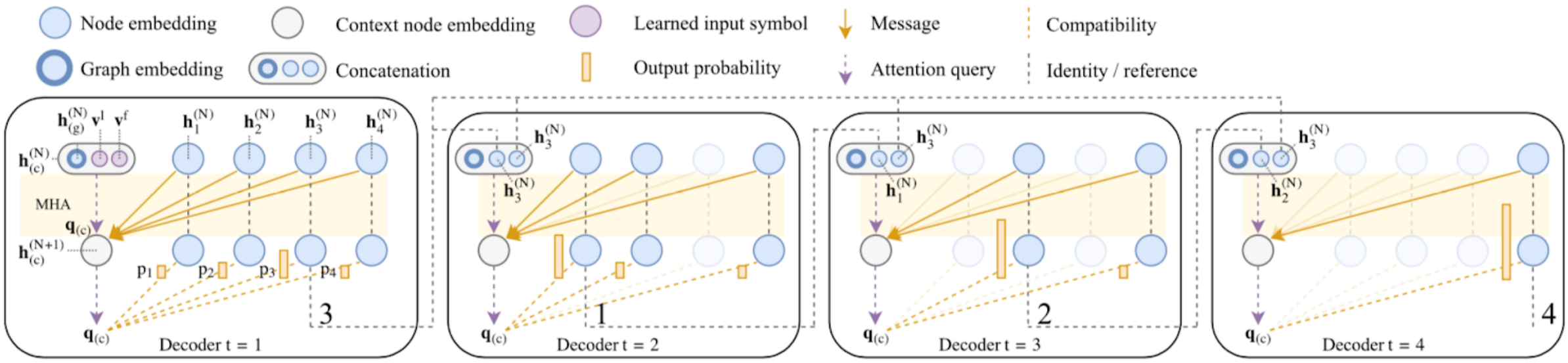


Figure 2: Attention based decoder for the TSP problem. The decoder takes as input the graph embedding and node embeddings. At each time step t, the context consist of the graph embedding and the embeddings of the first and last (previously output) node of the partial tour, where learned placeholders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

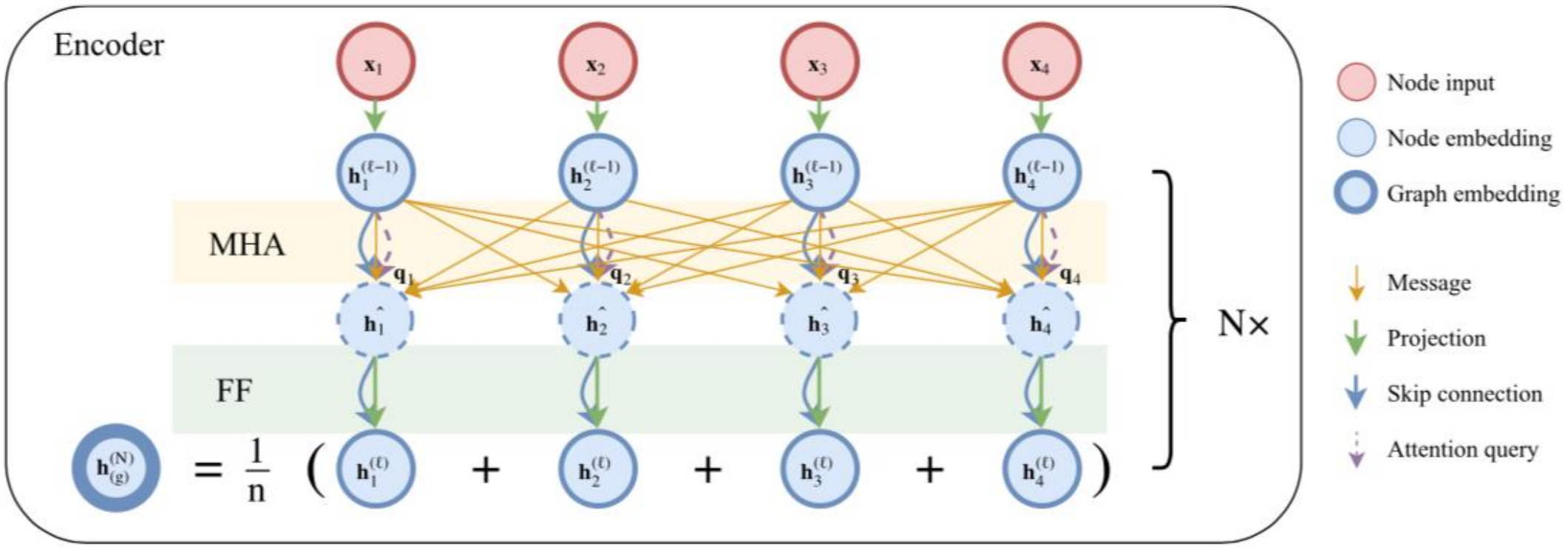
$$p_i = p_{\boldsymbol{\theta}}(\pi_t = i | s, \boldsymbol{\pi}_{1:t-1}) = \frac{e^{u_{(c)i}}}{\sum_j e^{u_{(c)j}}}.$$

$$\nabla \mathcal{L}(\boldsymbol{\theta}|s) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\pi}|s)} \left[(L(\boldsymbol{\pi}) - b(s)) \nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{\pi}|s) \right].$$

source: Wouter Kool, Herke van Hoof, Max Welling,
'Attention, Learn to Solve Routing Problems!', ICLR 2019.

A Simple Change in Model Architecture

We tried to modify encoder as putting soft-max attention over the layers by aiming to use all the previous layers. instead of the last layer.



However, no performance increase was achieved.

Experimental Design

Attention models are trained using data generated from 3 different distributions

- Uniform [0,1] (as implemented in the paper)
- Normal(μ = 0.5, σ = 0.25) scaled to [0,1]
- Exponential(λ = 1) scaled to [0, 1]

The results of the model are compared based on

- Deterministic greedy solution *- as implemented in the paper*
- Best solution among 1280 sampled solutions *- as implemented in the paper*
- Beam search with width 1280 *- as proposed but not implemented in the paper*

Baselines

- **Nearest Neighbor.** Start from a random node. In each iteration, select the nearest node to the last node. End at the same node.

- **Insertion.** Let S be the set of nodes in the partial tour. The next node i^* is selected as described below.

Nearest	Farthest	Random
$i^* = \arg\min_{i \notin S} \min_{j \in S} d_{ij}$	$i^* = \arg\max_{i \notin S} \min_{j \in S} d_{ij}.$	i* = a random node

- **LKH3.** a state-of-the-art heuristic solver that empirically also finds optimal solutions (Helsgaun, 2017)

Helsgaun, Keld. "An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems." *Roskilde: Roskilde University* (2017).

Test Data

Test data is generated using 3 different distributions
- 10k instances for each -

- Uniform [0,1] *- as implemented in the paper*
- Normal(μ = 0.5, σ = 0.25) scaled to [0,1]
- Exponential(λ = 1) scaled to [0, 1]

Results (TSP20)

Model		Test Data					
		Uniform		Normal		Exponential	
		Length	Gap	Length	Gap	Length	Gap
AM Greedy	Uniform	**3.847**	**0.29%**	4.023	0.51%	3.891	0.50%
	Normal	3.849	0.34%	**4.021**	**0.46%**	3.887	0.40%
	Exponential	3.866	0.80%	4.049	1.17%	**3.882**	**0.25%**
AM Sampling	Uniform	3.838	0.06%	4.007	0.11%	3.876	0.11%
	Normal	**3.838**	**0.06%**	**4.007**	**0.10%**	3.876	0.10%
	Exponential	3.842	0.16%	4.013	0.26%	**3.874**	**0.06%**
AM Beam	Uniform	3.836	0.00%	4.003	0.01%	3.872	0.01%
	Normal	**3.836**	**0.00%**	**4.003**	**0.01%**	3.872	0.01%
	Exponential	3.836	0.01%	4.005	0.05%	**3.872**	**0.00%**
Baselines	Nearest Neighbor	4.497	17.23%	4.768	19.12%	4.551	17.54%
	Farthest Insertion	3.926	2.36%	4.113	2.77%	3.933	1.59%
	Nearest Inserion	4.331	12.91%	4.452	11.23%	4.331	11.86%
	Random Insertion	4.003	4.36%	4.201	4.97%	4.029	4.07%
Optimal	LKH3	*3.836*	*0.00%*	*4.003*	*0.00%*	*3.872*	*0.00%*

Results (TSP50)

Model		Test Data					
		Uniform		Normal		Exponential	
		Length	Gap	Length	Gap	Length	Gap
AM Greedy	Uniform	**5.790**	**1.66%**	5.464	2.79%	4.971	3.59%
	Normal	5.839	2.52%	**5.457**	**2.65%**	4.950	3.16%
	Exponential	5.967	4.75%	5.921	11.37%	**4.858**	**1.24%**
Baselines	Nearest Neighbor	7.003	22.94%	6.573	23.65%	5.938	23.74%
	Farthest Insertion	6.011	5.53%	5.627	5.85%	4.993	4.06%
	Nearest Inserion	6.780	19.03%	6.243	17.43%	6.132	27.78%
	Random Insertion	6.132	7.65%	5.759	8.33%	5.112	6.53%
Optimal	LKH3	*5.696*	*0.00%*	*5.316*	*0.00%*	*4.799*	*0.00%*

Findings

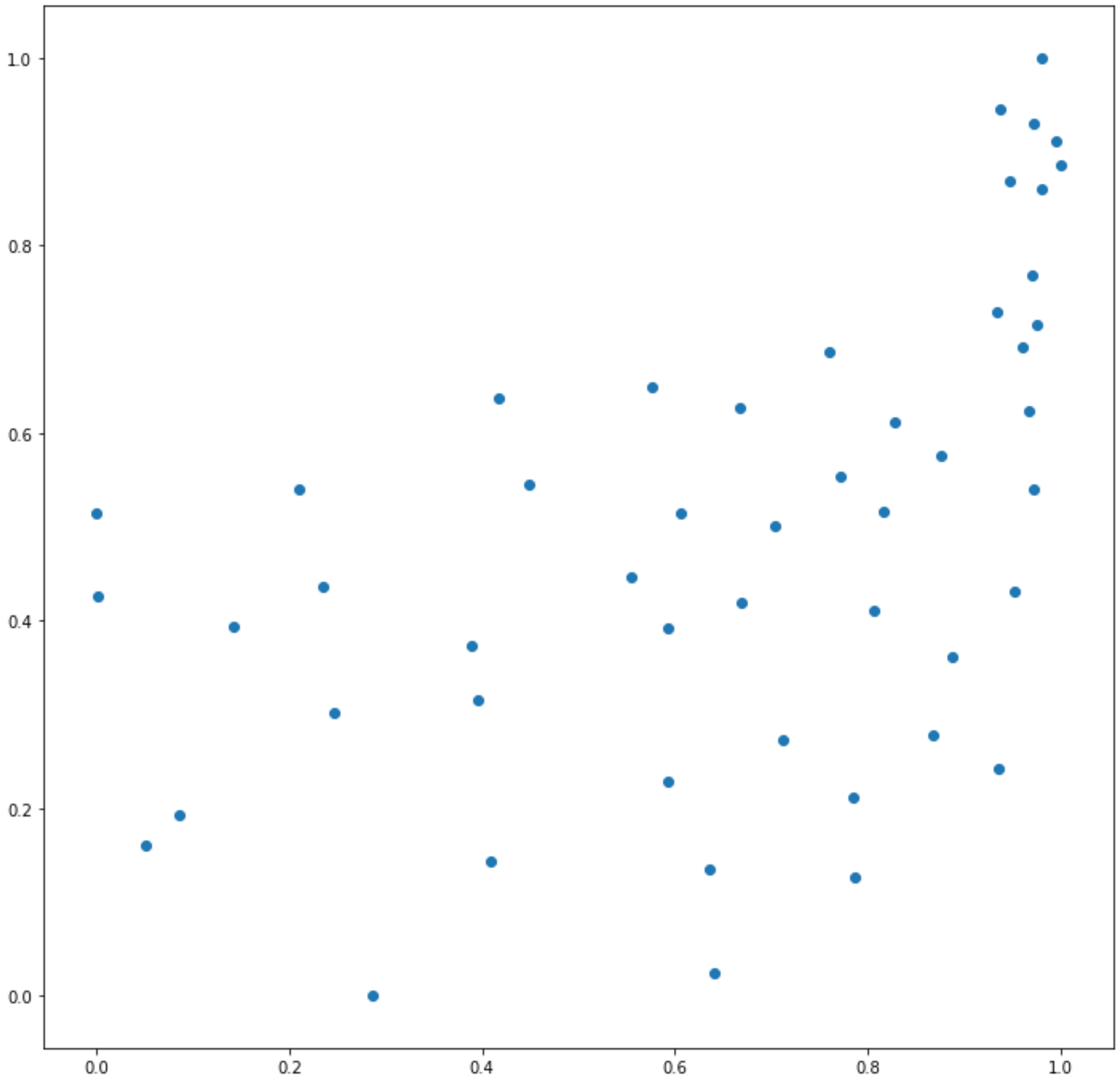
- The model should be trained with data from various distributions, or the known distribution of the test set.

- Beam search performs better than the random sampling, both time-wise and accuracy-wise.

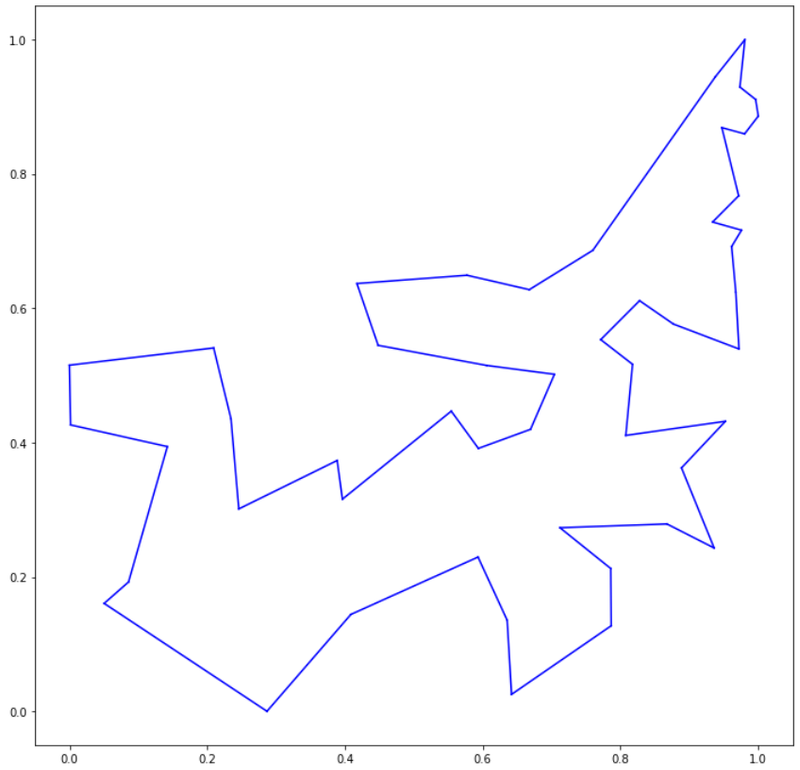
- The performance differences between models become larger as the size of problem instance grows.

- Currently, RL-based techniques cannot achieve the same accuracy as hand-crafted features for these small-sized instances. RL-based techniques could be more useful if improved to scale for larger instances.

Example

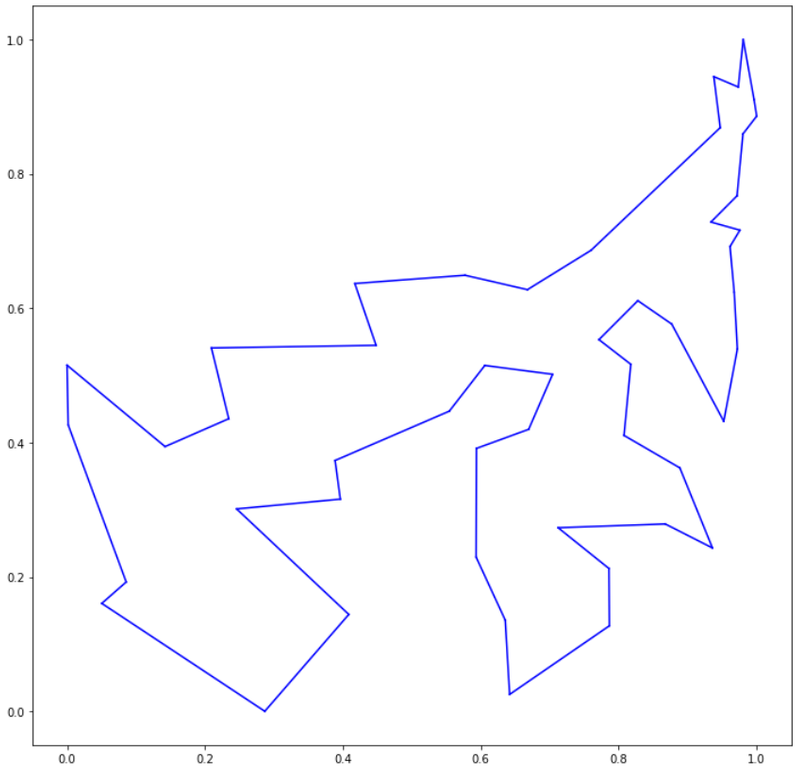


Solutions



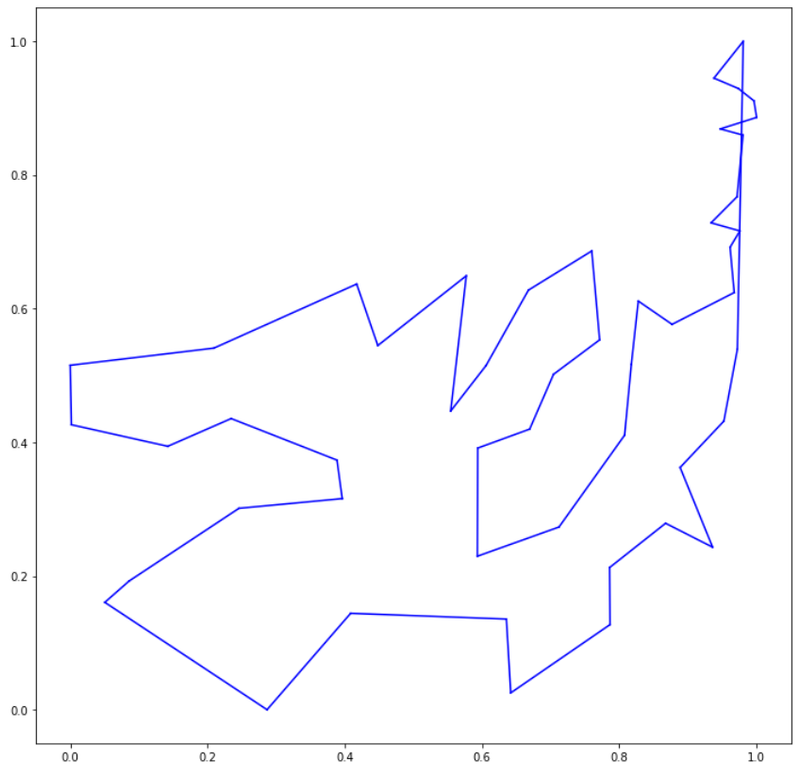
AM Greedy
uniform data

Length: 5.54



AM Greedy
normal data

Length: 5.66



AM Greedy
exponential data

Length: 5.98

Q&A

Thank you for listening!

- Furkan Gürsoy – Boğaziçi University
  furkangursoy.github.io

- Batuhan Koyuncu – Boğaziçi University
  bkoyuncu.github.io

Special thanks to Sercan Amaç – Hacettepe University.