(Reinforcement) Learning for Guiding Metaheuristics

Günther R. Raidl

Institute of Logic and Computation, TU Wien, Austria, raidl@ac.tuwien.ac.at

ELLIIT Hybrid AI Seminar
November 3, 2022
Combinatorial Optimization and Learning

- AI/machine learning boom also hit the area of combinatorial optimization
- This in many different ways

Focus here: utilize learning to better solve combinatorial optimization problems (COPs) in heuristic way

Basic idea of learning in MHs not new!
that the categories and their relationships to each other have been chosen carefully enough to indicate areas requiring research efforts as well as to help classify future work. We distinguish three hierarchical ways to use ML in metaheuristics (Fig. 1):

1. **Problem-level data-driven metaheuristics**: ML can help in modeling the optimization problem to solve (e.g., objective function, constraints). It can also assist landscape analysis and the decomposition of the problem.

2. **Low-level data-driven metaheuristics**: a metaheuristic is composed of different search components. ML can drive any search component such as the initialization of solution(s), and the search variation operators (e.g., neighborhoods in local search, mutation and crossover in evolutionary algorithms). It may also be used to tune the various parameters of a metaheuristic.

3. **High-level data-driven metaheuristics**: this class of data-driven metaheuristics concerns the selection and generation of metaheuristics, and the design of hybrid and parallel cooperative metaheuristics.

Other flat criteria are used in the taxonomy such as the learning time. In offline data-driven metaheuristics, the ML process occurs a priori before starting to solve the problem. In online data-driven metaheuristics, ML gathers knowledge during the search while solving the problem.

The synergy between ML and optimization has received increasing attention. Most of the related works basically focus on the use of optimization algorithms in solving ML problems [24][192][126][42][51]. Indeed, most of the ML problems can be formulated as optimization problems.

In the last decade, there was considerable interest in the use of ML into optimization. Very few papers investigate the role of ML into exact optimization algorithms (e.g., branch and bound, dynamic programming), constraint programming, and mathematical programming [20]. To our knowledge, there is no comprehensive survey which identifies in a unified way how ML can help the design of metaheuristics. In some outdated surveys [100][42][234], the authors enumerate some data-driven metaheuristics. In [33], the authors focus on dynamic combinatorial optimization problems. In [196], we have proposed a taxonomy of hybrid metaheuristics, in which the combination of metaheuristics with mathematical programming, constraint programming, and ML has been addressed. In this (from Talbi (2021))
Reinforcement Learning (RL)

- A sub-discipline of machine learning
- Environment is usually considered a Markov decision process
- Framework:

  ![Diagram of reinforcement learning cycle]

  - Environment
  - Agent
  - Action
  - State observed
  - Reward

Constructing a solution to a COP can be seen as an episode in an environment, objective value $\hat{\text{reward}}$.
Reinforcement Learning (RL)

- A sub-discipline of machine learning
- Environment is usually considered a Markov decision process
- Framework:

  ![Diagram of Reinforcement Learning](image)

  - Environment
  - Agent
  - Action
  - State observed
  - Reward

- Constructing a solution to a COP can be seen as an episode in an environment, objective value \( \hat{\text{reward}} \)
Reinforcement Learning (RL) - Classification

(from Mazyavkina et al. (2021))
Encoding of Problems+States, ML Models

- encoding highly problem-specific

- variants of (deep) neural networks dominate the used ML models
  - recurrent neural networks, e.g., LSTMs
  - pointer networks (Vinyals et al., 2015)
  - variants of Graph Neural Networks (Scarselli et al., 2008), e.g.,
    - Structure-to-Vector Network (Dai et al., 2016)
    - Graph Convolutional Network (Kipf and Welling, 2017)
    - Graph Isomorphism Network (Xu et al., 2019)
    - Graph Attention Network (Kool et al., 2019; Joshi et al., 2021)
Learning to Solve Graph Problems

- Dai et al. (2017): S2V-DQN
- min vertex cover, max cut, TSP considered
- graph embedding network structure2vec used to “featurize” nodes
- variant of Q-learning used to obtain a policy for greedily constructing solutions

![Diagram showing 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).]

Despite the inherent similarity between problem instances arising in the same domain, classical algorithms do not systematically exploit this fact. However, in industrial settings, a company may be willing to invest in upfront, offline computation and learning if such a process can speed up its real-time decision-making and improve its quality. This motivates the main problem we address:

**Problem Statement:**
Given a graph optimization problem $G$ and a distribution $D$ of problem instances, can we learn better heuristics that generalize to unseen instances from $D$?

Recently, there has been some seminal work on using deep architectures to learn heuristics for combinatorial problems, including the Traveling Salesman Problem \[37, 6, 14\]. However, the architectures used in these works are generic, not yet effectively reflecting the combinatorial structure of graph problems. As we show later, these architectures often require a huge number of instances in order to learn to generalize to new ones. Furthermore, existing works typically use the policy gradient for training \[6\], a method that is not particularly sample-efficient. While the methods in \[37, 6\] can be used on graphs with different sizes – a desirable trait – they require manual, ad-hoc input/output engineering to do so (e.g. padding with zeros).

In this paper, we address the challenge of learning algorithms for graph problems using a unique combination of reinforcement learning and graph embedding. The learned policy behaves like a meta-algorithm that incrementally constructs a solution, with the action being determined by a graph embedding network over the current state of the solution. More specifically, our proposed solution framework is different from previous work in the following aspects:

1. **Algorithm design pattern.** We will adopt a greedy meta-algorithm design, whereby a feasible solution is constructed by successive addition of nodes based on the graph structure, and is maintained so as to satisfy the problem’s graph constraints. Greedy algorithms are a popular pattern for designing approximation and heuristic algorithms for graph problems. As such, the same high-level design can be seamlessly used for different graph optimization problems.

2. **Algorithm representation.** We will use a graph embedding network, called structure2vec \[9\], to represent the policy in the greedy algorithm. This novel deep learning architecture over the instance graph “featurizes” the nodes in the graph, capturing the properties of a node in the context of its graph neighborhood. This allows the policy to discriminate among nodes based on their usefulness, and generalizes to problem instances of different sizes. This contrasts with recent approaches \[37, 6\] that adopt a graph-agnostic sequence-to-sequence mapping that does not fully exploit graph structure.

3. **Algorithm training.** We will use fitted Q-learning to learn a greedy policy that is parametrized by the graph embedding network. The framework is set up in such a way that the policy will aim to optimize the objective function of the original problem instance directly. The main advantage of this approach is that it can deal with delayed rewards, which here represent the remaining increase in objective function value obtained by the greedy algorithm, in a data-efficient way; in each step of the greedy algorithm, the graph embeddings are updated according to the partial solution to reflect new knowledge of the benefit of each node to the final objective value. In contrast, the policy gradient approach of \[6\] updates the model parameters only once w.r.t. the whole solution (e.g. the tour in TSP).
Learning to Solve Graph Problems (cont.)

- Kool et al. (2019)
- Autoregressive multi-head attention-based encoder/decoder GNN for TSP, VRP
- Trained with REINFORCE
Learning to Solve Graph Problems (cont.)

- Li et al. (2018)
- max independent set, min vertex cover, max clique, SAT considered
- **Graph Convolutional Network (GCN)** used to predict likelihood of each node to be part of a solution
- GCN yields **multiple probability maps** to account for the fact that multiple optimal solutions may exist
- **heuristic tree search** utilizing multiple maps, graph reduction, basic local search applied
- **supervised learning** instead of reinforcement learning learning
- results competitive to state-of-the-art solvers reported
Learning to Solve Graph Problems

- Abe et al. (2020): CombOptZero
- min vertex cover, max cut, max clique problems considered
- based on the principles of AlphaGoZero
- different graph neural networks tested, including GCN
- special reward normalization applied
- outperforms S2V-DQN, results close to state-of-the-art reported
Learning Beam Search (Huber and Raidl, 2021)

Randomly generated problem instance

Main BS with beam width $\beta$
(solves problem instance)

NBS calls from selected nodes
(generates $\alpha$ training data)

While performing main BS

ML model (e.g. NN)
(guides BS)

FIFO replay buffer of size $\gamma$
(stores training data, removes older samples)

<table>
<thead>
<tr>
<th>Feature Vectors</th>
<th>Targets</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,1]</td>
<td>27</td>
</tr>
<tr>
<td>[1,4]</td>
<td>24</td>
</tr>
<tr>
<td>[3,5]</td>
<td>22</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
A Learning Large Neighborhood Search for the Staff Rerostering Problem

F. Oberweger, G. Raidl, E. Rönnberg, and M. Huber
CPAIOR 22
Related Work

- Large Neighborhood Search (LNS) (Pisinger and Ropke, 2010)
- Decomposition-based learning LNS (Song et al., 2020)
- Neural LNS (Addanki et al., 2020)
- Neural Neighborhood Selection (NNS) (Sonnerat et al., 2021)

Our approach builds on NNS
Staff Rerostering Problem (SRRP)

- **Given:** old schedule, disruptions, demand to be met
- **Goal:** create new schedule
  - meeting new demand as best as possible (soft)
  - having as few changes to old schedule as possible (soft)
  - meeting all hard constraints, e.g., work regulations

![Figure: Overview of hard constraints.](image)
Large Neighborhood Search (LNS)

- Initial solution from a simple construction heuristic
- Repeated application of a destroy and a repair operators

- Repair: Mixed Integer Linear Programming (MILP) solver applied
Large Neighborhood Search (LNS)

- Initial solution from a simple construction heuristic
- Repeated application of a destroy and a repair operators

Repair: Mixed Integer Linear Programming (MILP) solver applied
Aiming to create a learning-based destroy operator
Repair Operator

- Regular MILP for feasible solutions
- MILP with relaxed hard constraints for infeasible solutions
  - Hard constraint violations are penalized
  - Objective value always worse for infeasible solution
Classical Randomized Destroy Operator

- Randomly choose employee-day pairs
- Destroy all variables associated with employee-day pairs

Figure: Destroy operator applied on an example SRRP instance.
Classical Randomized Destroy Operator

- Randomly choose employee-day pairs
- Destroy all variables associated with employee-day pairs

Figure: Destroy operator applied on an example SRRP instance.
Classical Randomized Destroy Operator

- **Consecutive day constraints**: selecting consec. days unlikely
- Better select and destroy random sequences of days!

![Destroy operator applied on an example SRRP instance.](image)

**Figure**: Destroy operator applied on an example SRRP instance.
Classical Randomized Destroy Operator

- **Consecutive day constraints:** selecting consec. days unlikely
- Better select and destroy *random sequences* of days!

![Figure: Destroy operator applied on an example SRRP instance.](image)

**Figure:** Destroy operator applied on an example SRRP instance.
Learning-Based Destroy Operator

Destroy Set Model

- Use Graph Neural Network (GNN) Scarselli et al. (2008)
- Model current solution as a graph in each state of LNS
- Predict weight of an employee-day pair to belong in destroy set

Figure: Simplified representation of the destroy set model architecture.
Learning-Based Destroy Operator

Destroy Set Model

- Use Graph Neural Network (GNN) Scarselli et al. (2008)
- Model current solution as a graph in each state of LNS
- Predict weight of an employee-day pair to belong in destroy set

Figure: Simplified representation of the destroy set model architecture.
Features

For each assignment \((n, d)\)

- flag indicating whether employee \(n\) is assigned to shift \(s \in S\) on day \(d\)
- flag indicating whether employee \(n\) is assigned to shift \(s \in S\) on day \(d\) in the original roster
- flag indicating whether employee \(n\) is absent on shift \(s \in S\) on day \(d\)
- flag indicating whether the minimum number of consecutive working days constraint is violated for employee \(n\) on day \(d\)
- flag indicating whether the maximum number of consecutive working days constraint is violated for employee \(n\) on day \(d\)
- flag indicating whether the minimum number of consecutive assignment constraint is violated for employee \(n\) on day \(d\) and shift \(s \in S\)
- flag indicating whether the maximum number of consecutive assignment constraint is violated for employee \(n\) on day \(d\) and shift \(s \in S\)
Features

For each employee $n$

- total number of working assignments of employee $n$
- total number of working assignments of employee $n$ minus minimum number of working days in the planning horizon ($\alpha_{\text{min}}$)
- maximum number of working days in the planning horizon ($\alpha_{\text{max}}$) minus total number of working assignments of employee $n$
- total number of assignments to shift $s \in S$ of employee $n$
- total number of assignments to shift $s \in S$ of employee $n$ minus minimum allowed number of assignments to this shift $s$ ($\gamma_{s\text{min}}$)
- maximum allowed number of assignments to shift $s \in S$ ($\gamma_{s\text{max}}$) minus total number of assignments to this shift $s$ of employee $n$
- total number of whole day absences of employee $n$
- total number of absences per shift $s \in S$ of employee $n$

For each Day $d$

- total number of assignments to each shift $s \in S$ on day $d$
- total number of assignments to each shift $s \in S$ on day $d$ minus cover requirements for this shift $s$ on day $d$ ($R_{ds}^c$)
Learning-Based Destroy Operator

Destroy Set Sampling Strategy

- Based on consecutive day observation
- Use GNN outputs $\mu_{nd}$ $\forall n \in N, d \in D$ for refined sampling

\[
\sum_{d_1}^{d_7} n_1 = 0.5 \\
\sum_{d_1}^{d_7} n_2 = 0.2 \\
\sum_{d_1}^{d_7} n_3 = 0.8
\]

**Figure:** Destroy set sampling strategy.
Learning-Based Destroy Operator

Destroy Set Sampling Strategy

- Based on consecutive day observation
- Use GNN outputs $\mu_{nd} \forall n \in N, d \in D$ for refined sampling

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
<th>$d_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>0.1</td>
<td>0.2</td>
<td>0.2</td>
<td>0.5</td>
<td>0.7</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>$n_2$</td>
<td>0.3</td>
<td>0.1</td>
<td>0.2</td>
<td>0.8</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>$n_3$</td>
<td>0.6</td>
<td>0.7</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
<td>0.5</td>
<td>0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
<th>$d_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
</tr>
<tr>
<td>$n_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure: Destroy set sampling strategy.
Learning-Based Destroy Operator

Destroy Set Sampling Strategy

- Based on consecutive day observation
- Use GNN outputs $\mu_{nd}$ $\forall n \in N, d \in D$ for refined sampling

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
<th>$d_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>0.1</td>
<td>0.2</td>
<td>0.2</td>
<td>0.5</td>
<td>0.7</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>$n_2$</td>
<td>0.3</td>
<td>0.1</td>
<td>0.2</td>
<td>0.8</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>$n_3$</td>
<td>0.6</td>
<td>0.7</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
<td>0.5</td>
<td>0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
<th>$d_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>0.3</td>
<td>0.5</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure:** Destroy set sampling strategy.
Learning-Based Destroy Operator

Destroy Set Sampling Strategy

- Based on consecutive day observation
- Use GNN outputs \( \mu_{nd} \forall n \in N, d \in D \) for refined sampling

<table>
<thead>
<tr>
<th></th>
<th>( d_1 )</th>
<th>( d_2 )</th>
<th>( d_3 )</th>
<th>( d_4 )</th>
<th>( d_5 )</th>
<th>( d_6 )</th>
<th>( d_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_1 )</td>
<td>0.1</td>
<td>0.2</td>
<td>0.2</td>
<td>0.5</td>
<td>0.7</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>( n_2 )</td>
<td>0.3</td>
<td>0.1</td>
<td>0.2</td>
<td>0.8</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>( n_3 )</td>
<td>0.6</td>
<td>0.7</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
<td>0.5</td>
<td>0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>( d_1 )</th>
<th>( d_2 )</th>
<th>( d_3 )</th>
<th>( d_4 )</th>
<th>( d_5 )</th>
<th>( d_6 )</th>
<th>( d_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_1 )</td>
<td>0.3</td>
<td>0.5</td>
<td>0.9</td>
<td>1.4</td>
<td>1.6</td>
<td>1.3</td>
<td>0.6</td>
</tr>
<tr>
<td>( n_2 )</td>
<td>0.4</td>
<td>0.6</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>( n_3 )</td>
<td>1.3</td>
<td>1.5</td>
<td>1.0</td>
<td>0.4</td>
<td>0.7</td>
<td>0.9</td>
<td>0.8</td>
</tr>
</tbody>
</table>

**Figure:** Destroy set sampling strategy.
Learning-Based Destroy Operator
Destroy Set Sampling Strategy

- Based on consecutive day observation
- Use GNN outputs $\mu_{nd} \forall n \in N, d \in D$ for refined sampling

random selection
proportional to weights

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
<th>$d_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>0.3</td>
<td>0.5</td>
<td>0.9</td>
<td>1.4</td>
<td>1.6</td>
<td>1.3</td>
<td>0.6</td>
</tr>
<tr>
<td>$n_2$</td>
<td>0.4</td>
<td>0.6</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>$n_3$</td>
<td>1.3</td>
<td>1.5</td>
<td>1.0</td>
<td>0.4</td>
<td>0.7</td>
<td>0.9</td>
<td>0.8</td>
</tr>
</tbody>
</table>

**Figure:** Destroy set sampling strategy.
Learning-Based Destroy Operator

Destroy Set Sampling Strategy

- Based on consecutive day observation
- Use GNN outputs $\mu_{nd}$ $\forall n \in N, d \in D$ for refined sampling

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
<th>$d_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>0.3</td>
<td>0.5</td>
<td>0.9</td>
<td>1.4</td>
<td>1.6</td>
<td>1.3</td>
<td>0.6</td>
</tr>
<tr>
<td>$n_2$</td>
<td>0.4</td>
<td>0.6</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>$n_3$</td>
<td>1.3</td>
<td>1.5</td>
<td>1.0</td>
<td>0.4</td>
<td>0.7</td>
<td>0.9</td>
<td>0.8</td>
</tr>
</tbody>
</table>

**Figure:** Destroy set sampling strategy.
Learning-Based Destroy Operator

Destroy Set Sampling Strategy

- Based on consecutive day observation
- Use GNN outputs $\mu_{nd} \ \forall n \in N, d \in D$ for refined sampling

update underlying weights

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
<th>$d_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>0.3</td>
<td>0.5</td>
<td>0.4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
</tr>
<tr>
<td>$n_2$</td>
<td>0.4</td>
<td>0.6</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>$n_3$</td>
<td>1.3</td>
<td>1.5</td>
<td>1.0</td>
<td>0.4</td>
<td>0.7</td>
<td>0.9</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Figure: Destroy set sampling strategy.
Learning-Based Destroy Operator

Destroy Set Sampling Strategy

- Based on consecutive day observation
- Use GNN outputs $\mu_{nd}$ $\forall n \in N, d \in D$ for refined sampling

<table>
<thead>
<tr>
<th>$n_1$</th>
<th>0.3</th>
<th>0.5</th>
<th>0.4</th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
<th>0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_2$</td>
<td>0.4</td>
<td>0.6</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>$n_3$</td>
<td>1.3</td>
<td>1.5</td>
<td>1.0</td>
<td>0.4</td>
<td>0.7</td>
<td>0.9</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Figure: Destroy set sampling strategy.
Learning-Based Destroy Operator

Destroy Set Sampling Strategy

- Based on consecutive day observation
- Use GNN outputs $\mu_{nd}$ $\forall n \in N, d \in D$ for refined sampling

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
<th>$d_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>0.3</td>
<td>0.5</td>
<td>0.4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
</tr>
<tr>
<td>$n_2$</td>
<td>0.4</td>
<td>0.6</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>$n_3$</td>
<td>1.3</td>
<td>1.5</td>
<td>1.0</td>
<td>0.4</td>
<td>0.7</td>
<td>0.9</td>
<td>0.8</td>
</tr>
</tbody>
</table>

**Figure:** Destroy set sampling strategy.

- Regulate influence of GNN with temperature $\tau$
  - Such that $\mu_{\frac{1}{\tau}nd}$ $\forall n \in N, d \in D$
  - So far $\tau = 1$
Learning-Based Destroy Operator

Temperature Model

- **Learn** temperature $\tau$ for each state with a GNN
- **Input:**
  - graph representation of current solution
  - destroy set model outputs
- **Output:** probabilities for selecting temperature in $\mathcal{T} = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3, 5\}$

![Diagram]

**Figure:** Simplified representation of the temperature model architecture.
Learning-Based Destroy Operator

Training

- Offline with representative problem instances via imitation learning

- **Expert policy:**
  MILP with local branching constraint to determine optimal destroy set

- **Loss function:** log-likelihood of expert actions, cross-entropy for temperature

- **DAGGER (Ross et al., 2011):**
  Trajectories are first created with expert strategy, later with learned model
Computational Results

- Model trained with $|N| = 110$
- MILP + Gurobi optimality gap between 26% and 34%

**Figure:** Comparison of LNS_RND and LNS_NN optimality gaps. 15 minutes running time. Lower bounds from solving MILP for three hours.
Conclusions

- Large variety of ML-based approaches to support/improve metaheuristics
- Modern RL techniques seem particularly promising
  - to reduce effort in manually crafting/tuning heuristics
  - without labeled training data (supervised learning)
- Naive application of an RL agent to a COP usually not competitive
- Combinations with tree search, local search and problem-specific heuristics can boost performance substantially
- Keep in mind:
  - (deep) neural networks not always necessary, e.g., other ML models may be faster & more robust
  - deep RL can be tricky


