Is Deep Reinforcement Learning ready for solving industrial optimization problems?

Yingqian Zhang (TU/e)

ACP Summer School 2023, Leuven, 13 July 2023
What I cover today

- A very brief introduction to DRL
- Example 1: learning to make online decisions (in collaborative picking)
- Example 2: (hybrid) learning to schedule jobs in manufacturing
Intro to (deep) reinforcement learning
agent
agent

environment

Credit:
Zaharah Bukhsh
agent

observations

environment

Credit:
Zaharah Bukhsh
agent

actions

observations

environment

Credit:
Zaharah Bukhsh
agent

environment

actions

rewards

observations

Credit: Zaharah Bukhsh
• Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal.
• Trial-and-error search
• Delayed reward

Credit: Zaharah Bukhsh
At each step $t$ the agent:
- Executes action $A_t$
- Receives observation $S_t$
- Receives scalar reward $R_t$

The environment:
- Receives action $A_t$
- Emits observation $S_{t+1}$
- Emits scalar reward $R_{t+1}$

Agent–environment interaction in a Markov decision process

$t$ increments after interaction with the environment
“RL is a computational approach to understanding and automating goal-directed learning and decision making. It is distinguished from other computational approaches by its emphasis on learning by an agent from direct interaction with its environment, without requiring exemplary supervision or complete models of the environment” (Sutton and Barto, 2015).

- Formalism for learning decision-making and control from experience
- Framework for learning to solve sequential decision-making problem
What is “deep” RL?

Deep Reinforcement Learning

Reinforcement Learning

Deep Learning

Deep Reinforcement Learning
A non-exhaustive taxonomy of algorithms in modern RL

Source: spinningup.openai.com
Success stories of DRL
Human-level control through DRL (1/4)

- 49 Atari games
- From pixel to actions (no domain knowledge)
- The change in score is the reward.
- Same algorithm.
- Same function approximator w/3M free parameters.
- Same hyperparameters
- Roughly human-level performance on 29 out 49 games.

Objective is to surround more territory than the opponent.

Vastly large state and action space.

Policy network predicts the move.

Value network predicts the winner of the game.

Monte-carlo tree search to narrow down the search space.

Several iterations – Self-play.


**AlphaGo** beats Go master Lee Se-dol (12 March 2016)

**Step 1**
Collect demonstration data and train a supervised policy.

- A prompt is sampled from our prompt dataset.
- A labeler demonstrates the desired output behavior.
- This data is used to fine-tune GPT-3.5 with supervised learning.

**Step 2**
Collect comparison data and train a reward model.

- A prompt and several model outputs are sampled.
- A labeler ranks the outputs from best to worst.
- This data is used to train our reward model.

**Step 3**
Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

- A new prompt is sampled from the dataset.
- The PPO model is initialized from the supervised policy.
- The policy generates an output.
- The reward model calculates a reward for the output.
- The reward is used to update the policy using PPO.

https://openai.com/blog/chatgpt/
Solving combinatorial optimization problems (4/4?)

source: NS

source: Wefabricate

source: vanderlande
Real-world optimization problems

(Combinatorial) optimization problems:
with objectives, constraints

- Route planning for package delivery
  - Vehicle routing problem (VRP)
- Stacking/packing problems in harbours/warehouses;
  Train shunting
  - (3D) bin-packing problem
- Scheduling jobs on production lines
  - Job shop/machine scheduling problem
Real-world optimization problem

Two types of problems

1. Stochastic, sequential decision making problems: need quick, online decisions, e.g. ambulance dispatching, order batching in e-commerce warehouses

2. Full information available, although some executions could have randomness, e.g. (usually) package delivery, scheduling problems
Successful story 4.1
End-to-end DRL for Collaborative picking

Igor Smit
Luca Begnardi
Order Picking

Order Picking is a crucial component of warehouse operation

- Order batching
- Order releasing
- Picker routing: which retrieving tasks should be assigned to each picker in order to optimize a certain metric, such as total order tardiness

Traditional human-only order picking

- Humans handle all the work
- Highly inefficient: only 30% of time spent picking, on average.
A brief history of picking

1-to-1 Manual

1-to-1 Autonomous vehicles

1-to-Many Autonomous vehicles

Many-to-Many Autonomous vehicles

Collaborative Order Picking

Complexity
Orchestrating chaos
Collaborative Order Picking: Challenge

How to allocate pickers to AMRs (autonomous mobile robots)?

Solution requirements

• Online allocation/decision
• Handling uncertainty and congestion
• Handling different picker/AMR numbers, different warehouse sizes
Optimization problem

• Develop a ‘picker optimizer’ in human-robot collaborative picking

• Allocating human pickers to AMRs, such that the total picking time is minimized (i.e., max pick rate)

What we know

- Business rules are available
- Data (orders, locations of items) available
Environment: simulation model

Representing interactive processes between pickers, AMRs, and the picker optimizer.

Uncertainties
- stochastic picker/AMR speeds and picking times
- picking disruptions
- congestion causing delays for the AMRs related to overtaking procedures
DRL approach

We have built the environment

Next:

- MDP (Markov decision process) formulation
- State space (and embedding): most tricky part!
- Action space
- Reward function
Spatial input representation

Classical feed-forward neural networks not suitable as

• Cannot adapt to different problem instances due to fixed input size; hard to capture important spatial information

Instance as a graph

• Nodes: locations in warehouse; Edges: how to travel between locations
State Representation

Node features capture information for each node/location

1. Controlled picker
2. AMR information
3. Picker information
4. Node location information
5. Node neighborhood information
Reward Function

Action Space:

• New picker destination
• Truncated action space: current or next AMR destination where no other picker is going

• Reward function: Penalty on time that passes
  \[ R_t^{\text{efficiency}} = \tau_{t-1} - \tau_t \]
Agent architecture design

Network must handle graph space and extract spatial information

Regular graph neural networks $\rightarrow$ Message passing

We adapt Invariant Feed-Forward network (Alomrani et al., 2022), with aisle-embedding: capture spatial relations in warehouse, size agnostic
Learning Algorithm

Proximal Policy Optimization (PPO) (Schulman et al., 2017)

- Policy-based method: policy network directly outputs the actions

Actor-critic

- Actor network: suggests actions
- Critic network: estimates the advantage function (how good is the selected action)
Experiments
Experimental settings

• Performance of fixed warehouse sizes: picking time
• Picker/AMR transferability
• Warehouse size transferability

Multiple warehouse types

<table>
<thead>
<tr>
<th>Warehouse Type</th>
<th>Aisles</th>
<th>Aisle Depth</th>
<th>Locations</th>
<th>Pickers</th>
<th>AMRs</th>
<th>Picks</th>
</tr>
</thead>
<tbody>
<tr>
<td>XS</td>
<td>7</td>
<td>7</td>
<td>98</td>
<td>4</td>
<td>7</td>
<td>≤ 100</td>
</tr>
<tr>
<td>S</td>
<td>10</td>
<td>10</td>
<td>200</td>
<td>10</td>
<td>25</td>
<td>5000</td>
</tr>
<tr>
<td>M</td>
<td>15</td>
<td>15</td>
<td>450</td>
<td>20</td>
<td>50</td>
<td>7500</td>
</tr>
<tr>
<td>L</td>
<td>25</td>
<td>25</td>
<td>1250</td>
<td>30</td>
<td>90</td>
<td>7500</td>
</tr>
<tr>
<td>XL</td>
<td>35</td>
<td>40</td>
<td>2800</td>
<td>60</td>
<td>180</td>
<td>15000</td>
</tr>
</tbody>
</table>
Benchmark algorithms

- **VI benchmark**: rule-based method considers the distance of potential picks and also tries to spread the pickers across aisles.

- **Greedy**: assign a picker to the closest available location where an AMR is going, and no other picker is already going.

- **MILP**: assuming no uncertainties/randomness, no congestions, in XS environment

\[
\begin{align*}
\sum_{k \in K} A_{A,k} &= 1 \\
A_{A,k} - A_{A,k} &\leq 1 - (U_{A,k} + U_{A,k}) \\
A_{A,k} + A_{A,k} &\leq 1 + (U_{A,k} + U_{A,k}) \\
B_{p,k} &\geq \tau_{t,k} - M \cdot (1 - A_{A,k}) \\
\sum_{k \in K} B_{p,k} &\geq \sum_{k \in K} F_{p,k} + \tau_{a,k} \cdot U_{A,k} - M \cdot (1 - U_{A,k}) \\
F_{p,k} &\geq F_{p,k} - M \cdot (2 - A_{A,k} - a_{p,k}) \\
\sum_{k \in K} B_{p,k} &\geq \sum_{k \in K} F_{p,k} + \tau_{a,k} \cdot a_{p,k} \\
B_{p,k} &\geq B_{p,k} - M \cdot (2 - A_{A,k} - a_{p,k}) \\
F_{p,k} &= B_{p,k} + a_{p,k} \cdot a_{p,k}
\end{align*}
\]

\[\forall i \in N \quad (2.1)\]
\[\forall i, r \in N, i \neq r, k \in K \quad (2.2)\]
\[\forall i, r \in N, i \neq r, k \in K \quad (2.3)\]
\[\forall i, r \in N, i \neq r \quad (2.4)\]
\[\forall i, r \in N, i \neq r \quad (2.5)\]
\[\forall i, r \in N, i \neq r \quad (2.6)\]
\[\forall i, r \in N, i \neq r \quad (2.7)\]
\[\forall i, r \in N, i \neq r \quad (2.8)\]
\[\forall i, r \in N, i \neq r \quad (2.9)\]
\[\forall i, r \in N, r \in R \quad (2.10)\]
\[\forall i, r \in N, r \in R, k \in K \quad (2.11)\]
\[\forall i, r \in N, r \in R \quad (2.12)\]
Experiment 1: performance on deterministic, XS instances

Compared to MILP solver: small instances without randomness and overtaking penalty. Each AMR does 1 pickrun.

Gurobi: gaps after 20 hours (DRL: solves in milliseconds)
Experiment 1: performance with various sizes

Values indicate the average picking time over 100 evaluation episodes

<table>
<thead>
<tr>
<th>Warehouse</th>
<th>DRL Picking Time</th>
<th>%</th>
<th>Greedy Picking Time</th>
<th>%</th>
<th>VI Benchmark Picking Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>8586 ± 62</td>
<td>14.9</td>
<td>10619 ± 59</td>
<td>-5.3</td>
<td>10087 ± 58</td>
</tr>
<tr>
<td>M</td>
<td>8425 ± 46</td>
<td>21.0</td>
<td>11023 ± 58</td>
<td>-3.3</td>
<td>10669 ± 41</td>
</tr>
<tr>
<td>L</td>
<td>6540 ± 37</td>
<td>31.7</td>
<td>9823 ± 33</td>
<td>-2.7</td>
<td>9569 ± 61</td>
</tr>
<tr>
<td>XL</td>
<td>9010 ± 21</td>
<td>33.6</td>
<td>13972 ± 44</td>
<td>-3.0</td>
<td>13570 ± 72</td>
</tr>
</tbody>
</table>
Experiment 2: Picker/AMR Transferability

Trained with:
- 10/25 for S
- 60/180 for XL

The trained policies are applied to scenarios with different numbers of pickers and AMRs and their ratios.

(a) Warehouse type S.

<table>
<thead>
<tr>
<th>Pickers/AMRs</th>
<th>DRL Picking Time</th>
<th>Greedy Picking Time</th>
<th>VI Benchmark Picking Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/15</td>
<td>12825 ± 83</td>
<td>15166 ± 74</td>
<td>15472 ± 87</td>
</tr>
<tr>
<td>10/20</td>
<td>9206 ± 51</td>
<td>11274 ± 69</td>
<td>11420 ± 56</td>
</tr>
<tr>
<td>10/30</td>
<td>8221 ± 54</td>
<td>10283 ± 60</td>
<td>9447 ± 52</td>
</tr>
<tr>
<td>15/25</td>
<td>6737 ± 42</td>
<td>7994 ± 40</td>
<td>8583 ± 36</td>
</tr>
<tr>
<td>15/30</td>
<td>5930 ± 34</td>
<td>7804 ± 55</td>
<td>7879 ± 46</td>
</tr>
<tr>
<td>15/35</td>
<td>5938 ± 35</td>
<td>7550 ± 44</td>
<td>7121 ± 38</td>
</tr>
</tbody>
</table>

(d) Warehouse type XL.

<table>
<thead>
<tr>
<th>Pickers/AMRs</th>
<th>DRL Picking Time</th>
<th>Greedy Picking Time</th>
<th>VI Benchmark Picking Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>50/120</td>
<td>12028 ± 23</td>
<td>16816 ± 32</td>
<td>20142 ± 112</td>
</tr>
<tr>
<td>60/140</td>
<td>10150 ± 20</td>
<td>14312 ± 27</td>
<td>17118 ± 101</td>
</tr>
<tr>
<td>60/200</td>
<td>9009 ± 44</td>
<td>14293 ± 88</td>
<td>13979 ± 87</td>
</tr>
<tr>
<td>80/180</td>
<td>8106 ± 77</td>
<td>11343 ± 30</td>
<td>13275 ± 83</td>
</tr>
<tr>
<td>80/200</td>
<td>8011 ± 59</td>
<td>11765 ± 52</td>
<td>12571 ± 91</td>
</tr>
<tr>
<td>80/220</td>
<td>6947 ± 19</td>
<td>10799 ± 40</td>
<td>11877 ± 84</td>
</tr>
</tbody>
</table>
Experiment 3: Warehouse Size Transferability

- Trained policies on specific sized are tested on different sized instances

<table>
<thead>
<tr>
<th>Warehouse</th>
<th>Policy S</th>
<th>Policy M</th>
<th>Policy L</th>
<th>Policy XL</th>
<th>Greedy</th>
<th>VI Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>8586 ± 62</td>
<td>9190 ± 53</td>
<td>8875 ± 58</td>
<td>8986 ± 51</td>
<td>10619 ± 59</td>
<td>10087 ± 58</td>
</tr>
<tr>
<td>M</td>
<td>7931 ± 42</td>
<td>8425 ± 46</td>
<td>8064 ± 41</td>
<td>8220 ± 37</td>
<td>11023 ± 58</td>
<td>10669 ± 41</td>
</tr>
<tr>
<td>L</td>
<td>6877 ± 31</td>
<td>7190 ± 42</td>
<td>6540 ± 37</td>
<td>6877 ± 23</td>
<td>9823 ± 33</td>
<td>9569 ± 61</td>
</tr>
<tr>
<td>XL</td>
<td>9478 ± 20</td>
<td>11275 ± 33</td>
<td>8567 ± 24</td>
<td>9010 ± 21</td>
<td>13972 ± 44</td>
<td>13570 ± 72</td>
</tr>
</tbody>
</table>
Ablation study: architecture comparison

INV-FF: invariant feed-forward network without aisle embedding

GIN: Graph Isomorphism Network

GCN: Graph Convolutional Network

<table>
<thead>
<tr>
<th>Warehouse</th>
<th>INV-FF</th>
<th>AISLE-EMB</th>
<th>GIN</th>
<th>GCN</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>8689 ± 58</td>
<td>8586 ± 62</td>
<td>8869 ± 55</td>
<td>11677 ± 67</td>
</tr>
<tr>
<td>M</td>
<td>8628 ± 40</td>
<td>8425 ± 46</td>
<td>14151 ± 75</td>
<td>13851 ± 65</td>
</tr>
<tr>
<td>L</td>
<td>6602 ± 29</td>
<td>6540 ± 37</td>
<td>11723 ± 76</td>
<td>14419 ± 88</td>
</tr>
</tbody>
</table>

Picking performance

<table>
<thead>
<tr>
<th>Warehouse</th>
<th>INV-FF</th>
<th>AISLE-EMB</th>
<th>GIN</th>
<th>GCN</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>1.44 ± 0.03</td>
<td>2.16 ± 0.03</td>
<td>3.10 ± 0.03</td>
<td>6.02 ± 0.05</td>
</tr>
<tr>
<td>M</td>
<td>1.53 ± 0.03</td>
<td>2.22 ± 0.03</td>
<td>3.25 ± 0.03</td>
<td>6.28 ± 0.05</td>
</tr>
<tr>
<td>L</td>
<td>1.53 ± 0.03</td>
<td>2.41 ± 0.03</td>
<td>3.41 ± 0.04</td>
<td>6.65 ± 0.05</td>
</tr>
<tr>
<td>XL</td>
<td>1.73 ± 0.03</td>
<td>2.57 ± 0.04</td>
<td>3.77 ± 0.04</td>
<td>7.19 ± 0.06</td>
</tr>
</tbody>
</table>

Inference time (in milliseconds)
Humans are not robots
where is fairness?
Multi-objective DRL

- Fairness: Minimize standard deviation of carried product masses
- Add to state space workload fairness features
  - Node-specific information, distributional information
- Add to reward function
  - Penalty on increase in standard deviation
- Learning algorithm, adapted from the prediction-guided MORL (Xu et al., 2020)
  - A meta-policy approach, to present a non-dominated set showing the trade-offs
Experiment: multi-objective fixed warehouse sizes

Warehouse S

Warehouse M

Warehouse L
Experiment: multi-objective fixed warehouse sizes

Warehouse L
Inspecting Policy Behavior

VI Benchmark

Greedy

DRL
Policy analysis

Approximating policy behavior with a decision tree
Performance of DT

Approximating policy behaviour with a decision tree

<table>
<thead>
<tr>
<th>Policy</th>
<th>Tree-Best</th>
<th>DRL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PT</td>
<td>WF</td>
</tr>
<tr>
<td>1</td>
<td>16373 ± 139</td>
<td>41 ± 5</td>
</tr>
<tr>
<td>2</td>
<td>12347 ± 83</td>
<td>41 ± 4</td>
</tr>
<tr>
<td>3</td>
<td>9350 ± 65</td>
<td>81 ± 5</td>
</tr>
<tr>
<td>4</td>
<td>9505 ± 62</td>
<td>161 ± 10</td>
</tr>
<tr>
<td>5</td>
<td>9561 ± 77</td>
<td>197 ± 14</td>
</tr>
<tr>
<td>6</td>
<td>9480 ± 79</td>
<td>225 ± 12</td>
</tr>
<tr>
<td>Pure Performance</td>
<td>8785 ± 57</td>
<td>304 ± 21</td>
</tr>
<tr>
<td>Pure Fairness</td>
<td>19141 ± 110</td>
<td>173 ± 20</td>
</tr>
</tbody>
</table>

Figure 8.5: Performance evaluation of decision tree policies.
Conclusions

- Efficiency improvements +- 40%
  - Higher warehouse capacity
  - Lower picking costs
- Good trade-off between efficiency and fairness
  - Explicitly outline achievable trade-offs
  - Simultaneous improvement of picking times and workload fairness
  - For large warehouse policy with 23.6% efficiency and 92% fairness improvements
- Good transferability
Is DRL ready for dynamic, real-world sequential decision making problems?

Yes! It gives better performance than handcrafted heuristic rules, which also generalizes well

Recipe

- Build a good (discrete event) simulation model
- Representing problem instances well: good features, (size-agnostic, general) architecture
- Applying model-free/model based DRL algorithms
Is DRL ready for dynamic, real-world sequential decision making problems?

Other real-world dynamic sequential decision problems

- Travelling repairwomen problem: Da Costa et al., Policies for the dynamic traveling maintainer problem with alerts, European Journal of Operational Research, 2023
- Order batching problem: Beeks et al., Deep Reinforcement Learning for a Multi-Objective Online Order Batching Problem, ICAPS 2022
- Cals et al., Solving the online batching problem using deep reinforcement learning, CAIE 2021
- Train shunting problem: Peer et al., Shunting Trains with Deep Reinforcement Learning, IEEE SMC, 2018
Is DRL ready for dynamic, real-world sequential decision making problems?

It can be even better...

Challenges & opportunities

- autoRL (Afshar et al. 2022)
- Instance representation (Ya et al. 2023)
- Non-Deep RL (Vos & Verwer, 2023)
- Simulation to real-world
- Adoption
Successful story 4.2
Flexible Job-Shop Scheduling
Flexible Job Shop Scheduling Problem (FJSP)

- A set of $n$ jobs, $m$ machines
- Each job $j$ contains an ordered sequence of operations $O_{i,j}$
- Each $O_{i,j}$ must be performed by one of the machines compatible with $O_{i,j}$
- $O_{i,j+1}$ can only start after $O_{i,j}$ is completed.
- $O_{i,j}$ has processing time on specific machine $m$
- Each machine can only process one operation at a time

**Objective:** find a schedule that minimizes *makespan* (the time when all jobs have been processed)
FJSP in practice...

Jobs (i.e., orders)
- Quantity
- Deadline
- Material availability
- Operations (milling steps)
  - Duration
  - Setup

Machines
- Eligibility
- Tools
- Maintenance
- Operator

Objectives
- Makespan
- Operational Cost
  - Manual labor
  - Consumed Tools
  - Logistic movements
  - ...

Figure 3.1: Job-Shop Scheduling, adopted from Yamada and Nakano (1992)
Existing approaches for FJSP

- Mathematical optimization models; Constraint programming
- Dispatching rules: (e.g., shortest processing times (SPT))
- Heuristics: constructive, metaheuristics
- Reinforcement learning based approaches
  - End-to-end DRL (Song et al., 2022)
  - Hybrid approaches: parameter controls of evolutionary algorithms with DRL (Chen et al., 2020)
What is missing in the literature?

- A comparison study between e2e DRL and a hybrid approach on benchmark FJSP instances and a real-world instance

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>?</td>
</tr>
<tr>
<td>Runtime</td>
<td>?</td>
</tr>
<tr>
<td>Scalability</td>
<td>?</td>
</tr>
<tr>
<td>Robustness/generalization</td>
<td>?</td>
</tr>
</tbody>
</table>
Two approaches

1. Ours: Self-Learning Effective Genetic Algorithm (SLEGA)
2. End-to-End DRL, adapted from (Song et al. 2022)
End-to-end DRL
E2E-DRL approach

*Estimated Makespan

$T^* = 9.5$

$T^* = 11$

Action

- $M_3$

Reward

-1.5

Environment
E2E-DRL

Figure 4.8: E2E-DRL Framework, adopted from Song et al. (2022)
E2E-DRL: MDP formulation (adapted from Song et al. 2022)

**Actions:**
Select operation to machine allocation action, from set of eligible actions.

**Reward:**
Absolute penalty if expected makespan increases.
E2E-DRL: state representation

State space: represented as a heterogeneous graph

- Operation nodes:
  Status; #neighboring machine; Average or actual processing time; #unscheduled jobs; Potential or actual start time; Time until the release date...
- Machine nodes:
  #neighboring operations; available time; utilization...
- Arcs:
  processing time; sequence dependent setup time...

Figure 4.15: Heterogeneous Graph representation of FJSP. A dotted line means processable, while a solid line means scheduled. Adopted from Song et al. (2022)
State embedding

- GAT (graph attention network) is used for machine node embedding.
- The message-passing step: a machine node aggregates information from only the direct neighbors (eligible operations on that machine).
- The operation node is then embedded given the eligible machines, the previous node, the next node and the node itself.

PPO with actor-critic structure
E2E-DRL approach

*Estimated Makespan

\[ T_{c*} = 11 \]
A hybrid GA-DRL approach
Genetic algorithm

Credit: Robbert Reijnen
based on
https://www.generative-design.org/02-deeper-dive/02-04_genetic-algorithms
Solution presentation for FJSP

Chromosome: operator sequence & machine allocation components

- The operation sequence determines the order in which operations should be scheduled.
- The machine allocation determines on which an operation is scheduled.
  - $O_{11}$ is scheduled on M1
  - $O_{12}$ on M3 ...

→ This enables a representation that is always feasible for FJSP

Figure 4.2: Example encoding format of a job schedule.
Genetic Algorithm

Crossover

• Machine selection crossover → maintain certain machine allocations and fill remaining with other solution

• Operation sequence crossover → preserve relative scheduling random selected jobs and fill with other solution

Gen 0 – makespan: 50
Gen 2 – makespan 46
Gen 8 – makespan 44
Gen 50 – makespan 36
SLEGA: Online control of Genetic Algorithm with DRL

New generation

Mutation

Reproduction

Evaluation

Selection

Mutation parameters

Reproduction parameters
SLEGA: GA controlled with DRL

**MDP formulation**

**State:**
- quality of individuals (mean, max, standard deviation)
- number of generations left,
- stagnation count

**Actions:**
- individual mutation rate, crossover rate and gene mutation rate

**Reward:**
- Objective value increased in new generation
Experiments
## Experiments

<table>
<thead>
<tr>
<th>Experiment</th>
<th>FJSP type</th>
<th>Datasets</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Vanilla FJSP</td>
<td>Literature &amp; Custom</td>
<td>Makespan</td>
</tr>
<tr>
<td>2</td>
<td>FJSP + SDST</td>
<td>Literature &amp; Custom</td>
<td>Makespan</td>
</tr>
<tr>
<td>3</td>
<td>FJSP-WF (SDST, release date, night time)</td>
<td>Custom</td>
<td>Makespan</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Criteria</th>
<th>SLEGA</th>
<th>E2E-DRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Runtime</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Scalable</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Robust</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
### Dataset

#### Benchmark datasets

#### Real-world data (WF)

| Instance | $n$ | $m$ | $h_i$ | $|M_{i,k}|$ | $p_{i,k,j}$ | LB | UB | CP |
|----------|-----|-----|-------|-------------|-------------|-----|-----|-----|
| mk01     | 10  | 6   | [5, 7] | 3           | [1, 7]      | 36  | 39  | 40  |
| mk02     | 10  | 6   | [5, 7] | 6           | [1, 7]      | 24  | 26  | 27  |
| mk03     | 15  | 8   | 10    | 5           | [1, 2, 0]   | 204 | 204 | 204 |
| mk04     | 15  | 8   | [3,10]| 3           | [1, 1, 0]   | 48  | 60  | 60  |
| mk05     | 15  | 4   | [5,10]| 2           | [5, 1, 0]   | 168 | 172 | 174 |
| mk06     | 10  | 15  | 15    | 5           | [1, 1, 0]   | 33  | 58  | 59  |
| mk07     | 20  | 5   | 5     | 5           | [1, 2, 0]   | 133 | 139 | 143 |
| mk08     | 20  | 10  | [5,15]| 2           | [5, 2, 0]   | 523 | 523 | 523 |
| mk09     | 20  | 10  | [10,15]| 5          | [5, 2, 0]   | 299 | 307 | 307 |
| mk10     | 20  | 15  | [10,15]| 5          | [5, 2, 0]   | 165 | 197 | 214 |

Table 5.3: Brandimarte FJSP instances description

<table>
<thead>
<tr>
<th>Instance</th>
<th>$n$</th>
<th>$m$</th>
<th>$s_{i,j,k,l}$</th>
<th>Flexibility</th>
<th>LB</th>
<th>UB</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>[3, 8]</td>
<td>Total</td>
<td>66</td>
<td>66</td>
<td>66</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>[3,10]</td>
<td>Partial</td>
<td>107</td>
<td>107</td>
<td>107</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>[6,15]</td>
<td>Partial</td>
<td>212</td>
<td>221</td>
<td>221</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>[8,21]</td>
<td>Partial</td>
<td>331</td>
<td>355</td>
<td>355</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>[3, 6]</td>
<td>Total</td>
<td>107</td>
<td>119</td>
<td>119</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>[5, 18]</td>
<td>Partial</td>
<td>310</td>
<td>320</td>
<td>320</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>5</td>
<td>[8,23]</td>
<td>Total</td>
<td>397</td>
<td>397</td>
<td>397</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>4</td>
<td>[4,13]</td>
<td>Total</td>
<td>216</td>
<td>253</td>
<td>253</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
<td>[4,11]</td>
<td>Total</td>
<td>210</td>
<td>210</td>
<td>210</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>5</td>
<td>[10,28]</td>
<td>Partial</td>
<td>427</td>
<td>516</td>
<td>516</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>6</td>
<td>[8,24]</td>
<td>Partial</td>
<td>403</td>
<td>468</td>
<td>468</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>7</td>
<td>[9,26]</td>
<td>Partial</td>
<td>396</td>
<td>446</td>
<td>446</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>7</td>
<td>[9,30]</td>
<td>Partial</td>
<td>396</td>
<td>466</td>
<td>466</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>7</td>
<td>[10,31]</td>
<td>Partial</td>
<td>496</td>
<td>554</td>
<td>554</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>7</td>
<td>[10,30]</td>
<td>Partial</td>
<td>414</td>
<td>514</td>
<td>541</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>7</td>
<td>[10,30]</td>
<td>Partial</td>
<td>614</td>
<td>635</td>
<td>634</td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>7</td>
<td>[10,31]</td>
<td>Partial</td>
<td>764</td>
<td>879</td>
<td>931</td>
</tr>
<tr>
<td>18</td>
<td>9</td>
<td>8</td>
<td>[10,30]</td>
<td>Partial</td>
<td>764</td>
<td>884</td>
<td>884</td>
</tr>
<tr>
<td>19</td>
<td>11</td>
<td>8</td>
<td>[10,30]</td>
<td>Partial</td>
<td>807</td>
<td>1088</td>
<td>1070</td>
</tr>
<tr>
<td>20</td>
<td>12</td>
<td>8</td>
<td>[10,33]</td>
<td>Partial</td>
<td>944</td>
<td>1267</td>
<td>1208</td>
</tr>
</tbody>
</table>

Table 5.6: Fattahi dataset description
E2E-DRL and SLEGA: Training

Train both our SO-SLEGA and E2E-DRL approaches on 100 different FJSP instances.

We do so for 5 different FJSP sizes from sodata, 10x05, 15x10, 20x05, 20x10, and a mix of the instance sizes.
## Results - Experiment 1 – Vanilla FJSP

<table>
<thead>
<tr>
<th>Approach</th>
<th>mkdata</th>
<th>Gopt</th>
<th>t(s)</th>
<th>vdata</th>
<th>rdata</th>
<th>vdata</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>163.3</td>
<td>-</td>
<td>-</td>
<td>1005</td>
<td>923</td>
<td>807.9</td>
</tr>
<tr>
<td>SPT</td>
<td>283.2</td>
<td>73.48%</td>
<td>-</td>
<td>1305.35</td>
<td>1184.80</td>
<td>28.36%</td>
</tr>
<tr>
<td>MOR</td>
<td>202.3</td>
<td>23.89%</td>
<td>-</td>
<td>1211.2</td>
<td>1064.0</td>
<td>15.28%</td>
</tr>
<tr>
<td>MWKR</td>
<td>200.17</td>
<td>22.58%</td>
<td>-</td>
<td>1179.9</td>
<td>1046.2</td>
<td>13.35%</td>
</tr>
<tr>
<td>FIFO</td>
<td>206.1</td>
<td>26.20%</td>
<td>-</td>
<td>1255.46</td>
<td>1082.1</td>
<td>17.24%</td>
</tr>
<tr>
<td>RANDOM</td>
<td>637.5</td>
<td>290.39%</td>
<td>60.00</td>
<td>1225.9</td>
<td>1212.9</td>
<td>31.41%</td>
</tr>
<tr>
<td>GREEDY</td>
<td>484.9</td>
<td>196.94%</td>
<td>1.30</td>
<td>1290.5</td>
<td>1150.4</td>
<td>24.64%</td>
</tr>
<tr>
<td>SO-GA</td>
<td>192.2</td>
<td>17.09%</td>
<td>30.02</td>
<td>1144.6</td>
<td>1105.3</td>
<td>19.75%</td>
</tr>
</tbody>
</table>

### Heuristics rules

- Vanilla GA

### Model trained on different sized instances

<table>
<thead>
<tr>
<th>Test Sets</th>
<th>SLGA (R. Chen et al., 2020)</th>
<th>DRL-G</th>
<th>DRL-S</th>
<th>SO-SLEGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>10x10</td>
<td>200.1 22.54% 1.22 1193.1 18.72% 1049.7 13.73% 856.1 5.97%</td>
<td>220.1 34.78% 1.22 1269.5 26.32% 1125.1 21.90% 897.4 11.08%</td>
<td>199.3 22.05% 1.21 1192.5 18.66% 1046.2 13.35% 841.3 4.13%</td>
<td>198.0 21.25% 1.29 1218.0 21.19% 1066.3 14.44% 845.0 4.59%</td>
</tr>
</tbody>
</table>

Table 6.1: Results of different algorithms on literature test data.
GA vs SLEGA

(a) Best makespan

(b) Average makespan
Results - Experiment 1 – Vanilla FJSP

E2E-DRL and SO-SLEGA (both trained on 15x10 instances) on the cudata.
Results - Experiment 1 – Vanilla FJSP

<table>
<thead>
<tr>
<th>Instance</th>
<th>J-M Ratio</th>
<th>OR-Tools LB</th>
<th>Computation Time</th>
<th>Gopt OR-Tools</th>
<th>DRL-S 15x10</th>
<th>SO-SLEGA 15x10</th>
<th>SO-GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>15x80</td>
<td>0.2</td>
<td>99(^1)</td>
<td>30s</td>
<td>0% (99(^1))</td>
<td>21% (120)</td>
<td>0% (99(^1))</td>
<td>0% (99(^1))</td>
</tr>
<tr>
<td>80x50</td>
<td>1.6</td>
<td>115</td>
<td>6.5h</td>
<td>6% (122)</td>
<td>33% (153)</td>
<td>63% (188)</td>
<td>75% (203)</td>
</tr>
<tr>
<td>90x10</td>
<td>9.0</td>
<td>146</td>
<td>7.5h</td>
<td>314% (604)</td>
<td>371% (687)</td>
<td>336% (637)</td>
<td>476% (695)</td>
</tr>
</tbody>
</table>

\(^1\)Optimal solution found. \(^2\) DRL-S computes a solution within 1-60 seconds, whereas the (SLE)GAs take from 1-5 minutes.

Distribution of makespan for 50-job, 50-machine instances for DRL-S(15x10) and SO-SLEGA(15x10).
## Results - Experiment 2 - FJSP + SDSTs

- Solution quality
- Transferability

### Table 6.3: Results of different algorithms on SDST literature test data.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>FTdata</th>
<th>$C_{max}$</th>
<th>$G_{opt}$</th>
<th>$\bar{t}(s)$</th>
<th>Win Count</th>
<th>Average Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB$^2$</td>
<td>536.4</td>
<td>-</td>
<td>46.81%</td>
<td>0.73</td>
<td>0</td>
<td>8.85</td>
</tr>
<tr>
<td>MWKR</td>
<td>787.4</td>
<td>-</td>
<td>46.81%</td>
<td>0.73</td>
<td>0</td>
<td>3.75</td>
</tr>
<tr>
<td>RANDOM</td>
<td>638.6</td>
<td>-</td>
<td>19.06%</td>
<td>60.00</td>
<td>10</td>
<td>3.75</td>
</tr>
<tr>
<td>GREEDY</td>
<td>667.5</td>
<td>-</td>
<td>24.45%</td>
<td>0.92</td>
<td>0</td>
<td>7.30</td>
</tr>
<tr>
<td>SO-GA</td>
<td>607.6</td>
<td>-</td>
<td>13.28%</td>
<td>59.24</td>
<td>4</td>
<td>5.10</td>
</tr>
<tr>
<td>DRL-G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>5.25</td>
</tr>
<tr>
<td>DRL-S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>2.70</td>
</tr>
<tr>
<td>DRL-S$^1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>3.10</td>
</tr>
<tr>
<td>SO-SLEGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td>1.60</td>
</tr>
<tr>
<td>SO-SLEGA$^1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>1.20</td>
</tr>
</tbody>
</table>

$^1$Models trained on vanilla FJSP (Experiment 1), $^2$lower bound calculated using best makespan found over various algorithms.
Results - Experiment 3 - WFdata

We train the E2E-DRL and SO-SLEGa approaches on instance sets of different sizes.

- WFdata: Release dates, deadlines, night times

<table>
<thead>
<tr>
<th>Instances</th>
<th>Training</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E2E-DRL</td>
<td>SO-SLEGa</td>
</tr>
<tr>
<td>17x02</td>
<td>0.8h</td>
<td>3.3h</td>
</tr>
<tr>
<td>42x02</td>
<td>2.4h</td>
<td>6.2h</td>
</tr>
<tr>
<td>64x04</td>
<td>4.8h</td>
<td>7.5h</td>
</tr>
<tr>
<td>88x08</td>
<td>7.4h</td>
<td>10.9h</td>
</tr>
</tbody>
</table>

ACP Summer School 2023
Results - Experiment 3 - WFdata

WFdata: with additional constraints: release dates, deadlines, night times

Instances where the number of jobs ranges between 5 and 100 and the number of machines ranges between 2 and 10 are considered.

An instance of size 100x10 matches the industry scale.

<table>
<thead>
<tr>
<th>Instance</th>
<th>$C_{max}$</th>
<th>$G_{opt}$</th>
<th>$t(s)$</th>
<th>Win Count</th>
<th>Average Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB</td>
<td>132173</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MWKR</td>
<td>266848</td>
<td>101.89%</td>
<td>6.22</td>
<td>1</td>
<td>6.85</td>
</tr>
<tr>
<td>RANDOM</td>
<td>195869</td>
<td>46.95%</td>
<td>10.00</td>
<td>3</td>
<td>5.29</td>
</tr>
<tr>
<td>GREEDY</td>
<td>164040</td>
<td>23.07%</td>
<td>15.63</td>
<td>24</td>
<td>3.16</td>
</tr>
<tr>
<td>SO-GA</td>
<td>161651</td>
<td>22.31%</td>
<td>37.45</td>
<td>13</td>
<td>3.14</td>
</tr>
<tr>
<td>DRL-G</td>
<td>17x02</td>
<td>177421</td>
<td>34.23%</td>
<td>1.68</td>
<td>3.96</td>
</tr>
<tr>
<td></td>
<td>42x02</td>
<td>181551</td>
<td>37.44%</td>
<td>1.47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64x04</td>
<td>192354</td>
<td>45.53%</td>
<td>2.32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>88x08</td>
<td>187690</td>
<td>42.00%</td>
<td>2.20</td>
<td></td>
</tr>
<tr>
<td>DRL-S</td>
<td>17x02</td>
<td>175477</td>
<td>32.76%</td>
<td>5.46</td>
<td>3.51</td>
</tr>
<tr>
<td></td>
<td>42x02</td>
<td>182804</td>
<td>38.31%</td>
<td>4.35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64x04</td>
<td>177147</td>
<td>34.03%</td>
<td>2.87</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>88x08</td>
<td>183488</td>
<td>38.82%</td>
<td>3.35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15x10</td>
<td>188860</td>
<td>42.88%</td>
<td>2.88</td>
<td></td>
</tr>
<tr>
<td>SO-SLEGa</td>
<td>17x02</td>
<td>138733</td>
<td>4.96%</td>
<td>35.92</td>
<td>1.21</td>
</tr>
<tr>
<td></td>
<td>42x02</td>
<td>160726</td>
<td>21.60%</td>
<td>33.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64x04</td>
<td>142333</td>
<td>7.69%</td>
<td>29.65</td>
<td></td>
</tr>
<tr>
<td></td>
<td>88x08</td>
<td>144898</td>
<td>9.63%</td>
<td>30.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15x10</td>
<td>143992</td>
<td>8.03%</td>
<td>39.54</td>
<td></td>
</tr>
</tbody>
</table>

1Models trained on vanilla FJSP (Experiment 1), 2lower bound calculated using best makespan found on instance and all algorithms.

Table 6.7: Results of different algorithms on WF-specific instances.
Results - Experiment 3 - WFdata
When E2E-DRL falls short?

<table>
<thead>
<tr>
<th>Index</th>
<th>Instance Type</th>
<th>Average makespan</th>
<th>SO-SLEGA</th>
<th>DRL-S</th>
<th>DRL-S stack. feat.</th>
<th>DRL-S all feat.</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FJSP</td>
<td>48170</td>
<td>45051</td>
<td>45821</td>
<td>45821</td>
<td>46556</td>
<td>770 (1.7%)</td>
</tr>
<tr>
<td>2</td>
<td>(1) with SDSTs</td>
<td>102778</td>
<td>93992</td>
<td>119522</td>
<td>118148</td>
<td>128335</td>
<td>25530 (20.5%)</td>
</tr>
<tr>
<td>3</td>
<td>(2) with Release Dates</td>
<td>115135</td>
<td>103773</td>
<td>129749</td>
<td>126371</td>
<td>132535</td>
<td>22598 (17.9%)</td>
</tr>
<tr>
<td>4</td>
<td>(3) with Night Times</td>
<td>164040</td>
<td>138733</td>
<td>184627</td>
<td>175447</td>
<td>175447</td>
<td>36714 (20.9%)</td>
</tr>
<tr>
<td>5</td>
<td>(1) with Night Times</td>
<td>71760</td>
<td>64261</td>
<td>68276</td>
<td>67727</td>
<td>73552</td>
<td>3466 (5.4%)</td>
</tr>
</tbody>
</table>

Table 6.9: Ablation study for FJSP characteristics and E2E-DRL features.
# Hybrid approach vs E2E DRL

<table>
<thead>
<tr>
<th>Criteria</th>
<th>SLEGA</th>
<th>E2E-DRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Performs well generally speaking.</td>
<td>Fails to perform when under more complicated constraints and instances.</td>
</tr>
<tr>
<td>Runtime</td>
<td>Fast enough.</td>
<td>Almost instant.</td>
</tr>
<tr>
<td>Scalability</td>
<td>Generalizable to larger instances. Can be parallelized in order to support scalability.</td>
<td>Training on large instances does not work. Retraining is necessary for different instance characteristics.</td>
</tr>
<tr>
<td>Transferability/robustness</td>
<td>Can handle various instance types.</td>
<td>Fails to deal with complicated instance types.</td>
</tr>
</tbody>
</table>
Are DRLs there?

- End-to-end DRL
  - Fast (on solving)
  - Hard to scale, hard to handle heavily constrained problems
- Hybrid approaches work
  - learning to guide search/speed up solution finding
  - highly transferable/generalizable
### Coming soon: Job Shop Scheduling Benchmark

A open sourced repo for benchmarking scheduling solutions

- Benchmark instances of variants of JSP problems + various solution methods
- Environment of developing other (learning based) solution methods

**Release date V1: August 15, 2023:**
https://github.com/RobbertReijnen/Job_Shop_Scheduling_Benchmark

<table>
<thead>
<tr>
<th>Solutions methods</th>
<th>Job Shop</th>
<th>Flow Shop</th>
<th>Flexible</th>
<th>SDST</th>
<th>Assembly operations</th>
<th>Online Arrivals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heuristics</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Dispatching Rules</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GA: Genetic Algorithm</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SLEGA: GA with DRL</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>DRL – learning to dispatch</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>E2E DRL with GNN</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
The need of benchmarks

- The first AI for TSP competition @ IJCAI 2021
  https://tspcompetition.com/
  Zhang et al., The first AI4TSP competition: Learning to solve stochastic routing problems, Artificial Intelligence. 2023

- EURO Meets NeurIPS 2022 Vehicle Routing Competition,
  Kool et al., EURO Meets NeurIPS 2022 Vehicle Routing Competition. 2022
References

- Smit et al., Learning to Be Efficient and Fair for Collaborative Order Picking, working paper (happy to share)
- Begnardi et al., Deep Reinforcement Learning for Two-sided Online Bipartite Matching in Collaborative Order Picking, under review (happy to share)
- Van Straaten et al., Flexible Job-Shop Scheduling under heavy constraints with learning based approaches, working paper (happy to share)
- Zhang et al., The first AI4TSP competition: Learning to solve stochastic routing problems, Artificial Intelligence. 2023
- Kool et al., EURO Meets NeurIPS 2022 Vehicle Routing Competition. 2022
- Ya et al., Revisit the Algorithm Selection Problem for TSP with Spatial Information Enhanced Graph Neural Networks. 2023
- Da Costa et al., Policies for the dynamic traveling maintainer problem with alerts, European Journal of Operational Research, 2023
- Beeks et al., Deep Reinforcement Learning for a Multi-Objective Online Order Batching Problem, ICAPS 2022
- Cals et al., Solving the online batching problem using deep reinforcement learning, CAIE 2021
- Peer et al., Shunting Trains with Deep Reinforcement Learning, IEEE SMC, 2018
- Wu et al., Neural Airport Ground Handling, IEEE Transactions on Intelligent Transportation Systems, 2023
- Sutton and Barto, Reinforcement learning: an introduction
- Schulman et al. Proximal policy optimization algorithms, 2017
- Alomrani et al., Deep Policies for Online Bipartite Matching: A Reinforcement Learning Approach, TMLR, 2022
- Afshar et al., Automated Reinforcement Learning: An Overview. 2022
- Vos and Verwer, Optimal Decision Tree Policies for Markov Decision Processes, AAAI 2023
- Song et al., Flexible job shop scheduling via graph neural network and deep reinforcement learning. IEEE Transactions on Industrial Informatics, 2022
- Chen et al., A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. CAIE. 2020
Team AI for decision-making @TU/e