Allgemeine Informationen

- „Simulation and RL Framework for Production planning and control of complex job shop manufacturing systems“
  - **Sim-Model** erlaubt Parametrisierung diverser Job-Shop ähnlichen Fertigungssysteme
  - **RL** zur Steuerung des Auftragsbearbeitung
- Erlaubt Kombination und Nutzung diverser Deep RL Modelle (basieren auf Tensorforce)
- Beinhaltet Leistungsstatistiken und Logging
- **Nicht auf eigene Fabrik anwendbar → Eher zum Testen von Modellen**

Mögliche Konfigurationen

- **State** representation → Welche Zustände gibt es?
- **Reward** function → Gewichte von State / Action Paaren
- **Action** representation → Welche Aktionen sind ausführbar?
- Learning Rate, Discount Rate, Neural Network Config **in Tensorforce Agent**

Sources:
https://github.com/AndreasKuhnle/SimRLFab

Was beinhaltet das Simulationsmodell?

The simulation model covers the following features (`initialize_env.py`):

- Number of resources:
  - Machines: processing resources
  - Sources: resources where new jobs are created and placed into the system
  - Sinks: resources where finished jobs are placed
  - Transport resources: dispatching and transporting resources
- Layout of fix resources (machines, sources, and sinks) based on a distance matrix
- Sources:
  - Buffer capacity (only outbound buffer)
  - Job release / generation process
  - Restrict jobs that are released at a specific source
- Machines:
  - Buffer capacity (inbound and outbound buffers)
  - Process time and distribution for stochastic process times
  - Machine group definition (machines in the same group are able to perform the same process and are interchangeable)
  - Breakdown process based on mean time between failure (MTBL) and mean time of line (MTOL) definition
  - Changeover times for different product variants
- Sinks:
  - Buffer capacity (only inbound buffer)
- Transport resources:
  - Handling times
  - Transport speed
- Others:
  - Distribution of job variants
  - Handling times to load and unload resources
  - Export frequency of log-files
  - Turn on / off console printout for detailed report of simulation processes and debugging
  - Seed for random number streams

Sources:
https://github.com/AndreasKuhnle/SimRLFab

Allgemeine Informationen zum Entwurf

- Annäherung an optimale Policy durch Neural Network mit Policy Gradient (nicht Q-Learning)
- **Action Space Representation**
  - Aktionen werden durch Policy, basierend auf Gewichten des NN gewählt
- **State Representation**
  - Input Vector für NN
  - Im Gegensatz zum Action Space, muss State nicht klein gehalten werden
- **Reward Function**
  - Ziel ist kontinuierlich hohe Performance und kein optimaler State

Sources:
https://github.com/AndreasKuhnle/SimRLFab
https://reader.elsevier.com/reader/sd/pii/S22128..

Darstellung Funktionsweise

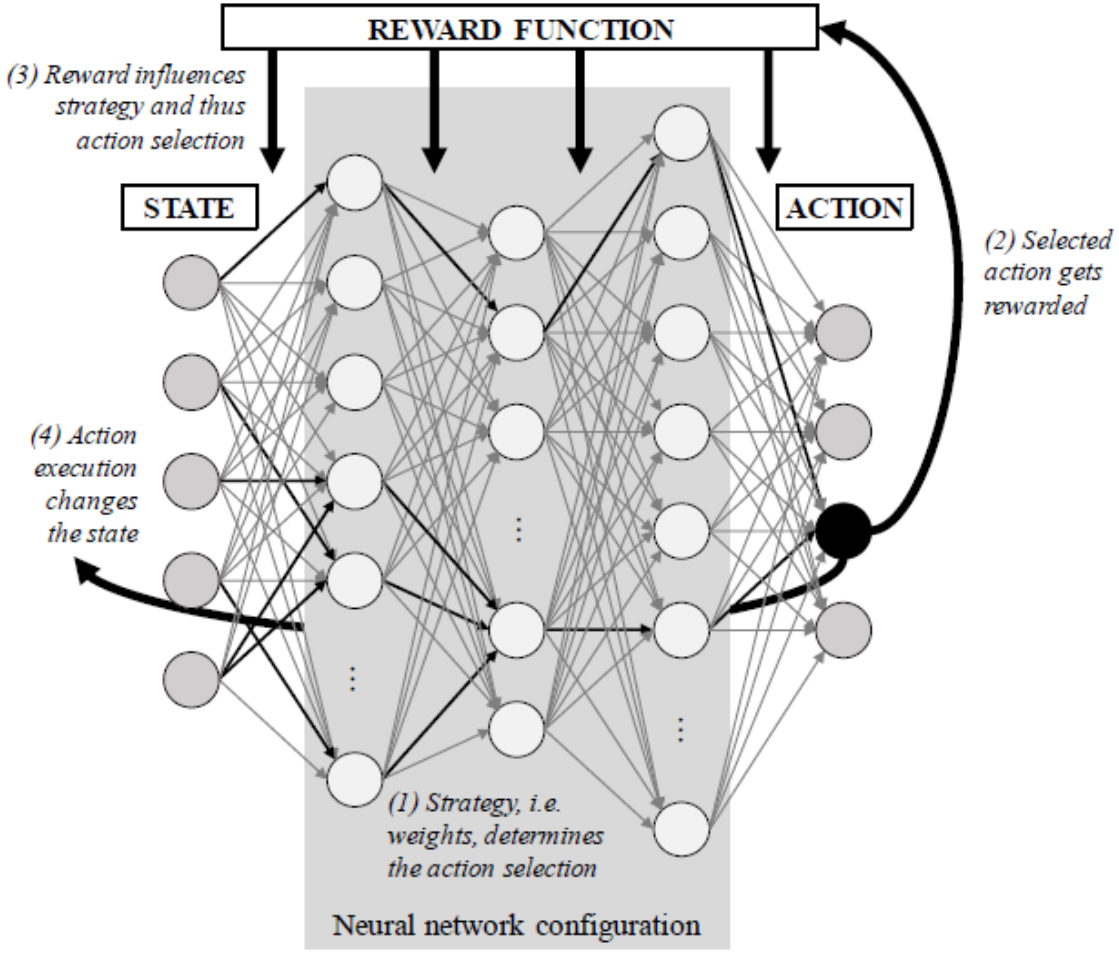


Fig. 2. ANN application, RL set-up, key elements and learning procedure.

Sources:
https://github.com/AndreasKuhnle/SimRLFab
https://reader.elsevier.com/reader/sd/pii/S22128..

Paper: Designing an adaptive production control system using reinforcement learning

- RL Agent soll optimale Auftragsdispositionsstrategie lernen
- RL Dispatching Agent besteht aus vier Modulen

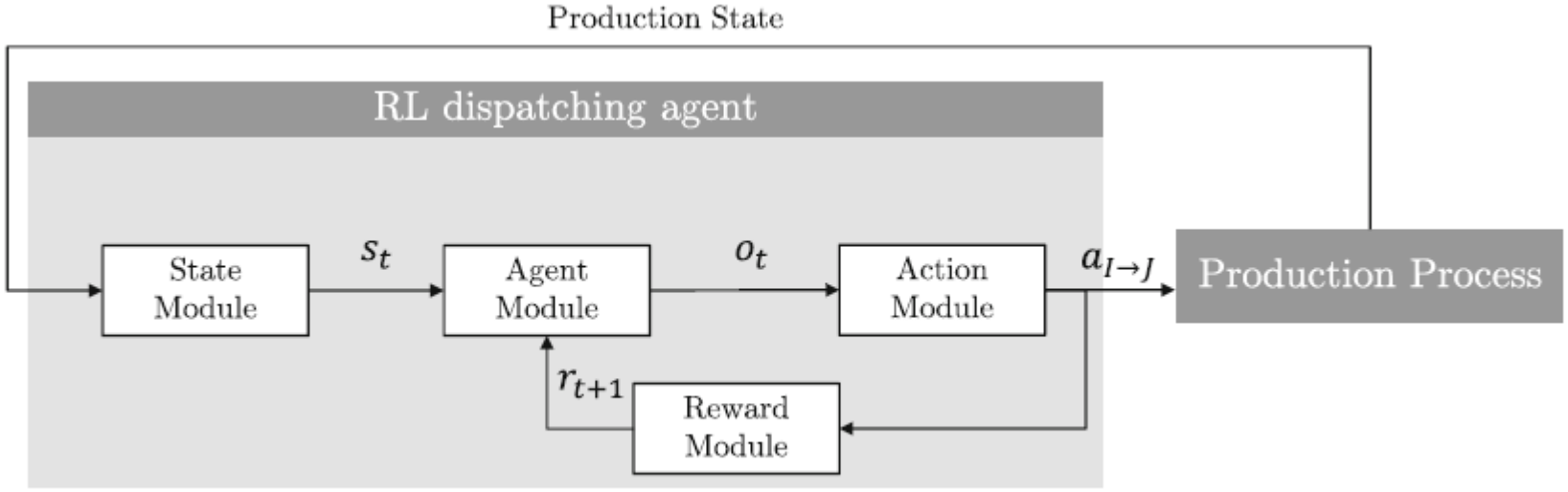
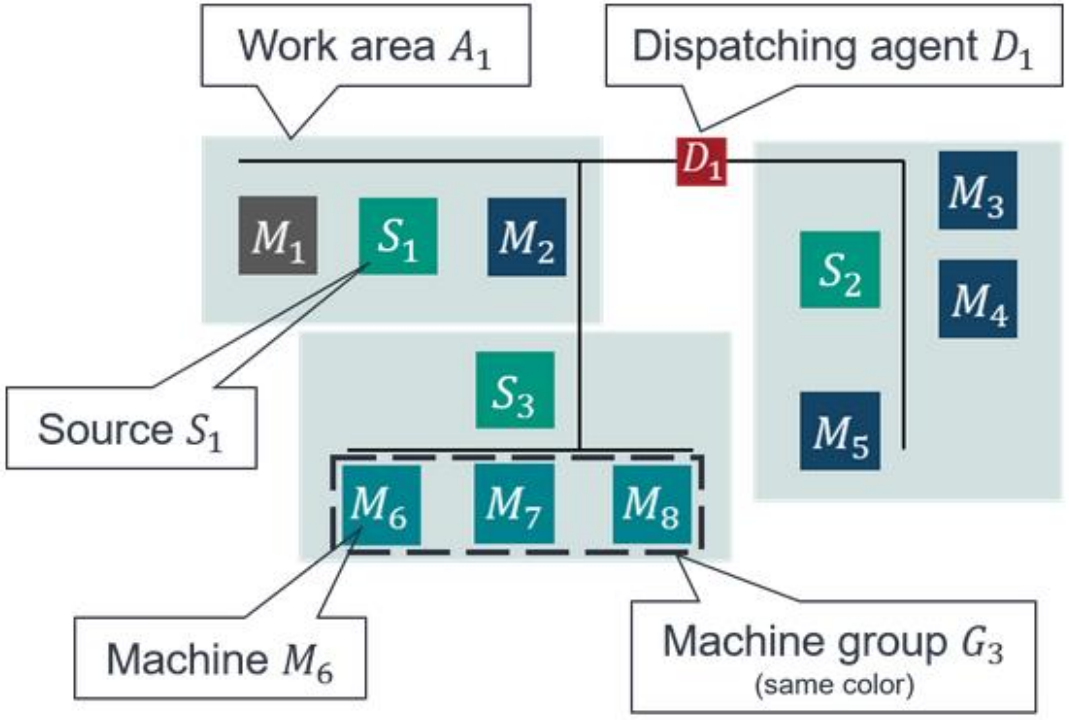


Fig. 3 Internal structure of the RL-agent with configurable sub-modules

Sources:
https://link.springer.com/article/10.1007/s10845-020-01612-y

Erläuterung des Simulationsbeispiels

- Wafer fabric with 8 machines, 3 sources and 3 sinks
- Dispatching Agent fährt durch Produktion



Sources:
https://link.springer.com/article/10.1007/s10845-020-01612-y

Erläuterung des Simulationsbeispiels

- Wafer fabric with 8 machines, 3 sources and 3 sinks
- Jede sink gehört zu einem Arbeitsbereich, denen Anzahl von Maschinen zugeordnet sind

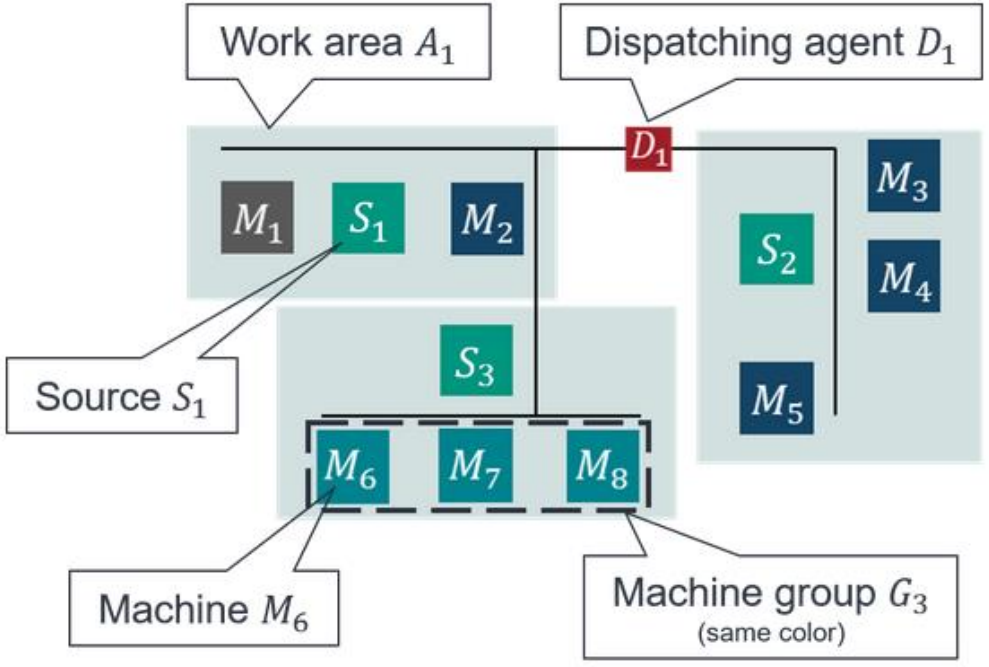


Table 3 Available action subsets according to the real-world use case

I \ J	$\varnothing$	$S_1 \ldots S_3$	$M_1 \ldots M_8$
$\varnothing$	$A_{waiting}$	A_{empty}	
$S_1 \ldots S_3$			$A_{S \to M}$
$M_1 \ldots M_8$		$A_{M \to S}$	

1. $A_{S \to M} = \{a_{S_1 \to M_1}, \ldots, a_{S_1 \to M_5}, a_{S_2 \to M_2}, \ldots, a_{S_2 \to M_5}, a_{S_3 \to M_6}, \ldots, a_{S_3 \to M_8}\}$ (12 actions)
2. $A_{M \to S} = \{a_{M_1 \to S_1}, a_{M_2 \to S_1}, a_{M_3 \to S_2}, a_{M_4 \to S_2}, a_{M_5 \to S_2}, a_{M_6 \to S_3}, a_{M_7 \to S_3}, a_{M_8 \to S_3}\}$ (8 actions)
3. $A_{empty} = \{a_{\varnothing \to S_1}, a_{\varnothing \to S_2}, a_{\varnothing \to S_3}, a_{\varnothing \to M_1}, \ldots, a_{\varnothing \to M_8}\}$ (11 actions)
4. $A_{waiting} = \{a_{\varnothing \to \varnothing}\}$ (1 action)

Sources:
https://link.springer.com/article/10.1007/s10845-020-01612-y

Erläuterung des Simulationsbeispiels

- RL Agent muss:
  - zwei Probleme in zwei Szenarien lösen (**Bestellreihenfolge** und **Routen Planung**)
  - **Zusammenspiel** zwischen **State-Informationen, ausgewählter Aktion** und **erhaltenen Reward lernen**, um **Performancewerte** zu optimieren
    - Maschinenauslastung, durchschnittliche Auftragswartezeit, durchschnittlicher Lagerbestand
- Trust Region Policy Optimization **(TRPO)** wird als Algorithmus verwendet
  - Verwendung von Grandientenanstieg zum Folgen der Policy mit dem höchsten Anstieg an Rewards
    - Mehr zum Algorithmus siehe https://jonathan-hui.medium.com/rl-trust-region-policy-optimization-trpo-explained-a6ee04eeeee9

Sources:
https://link.springer.com/article/10.1007/s10845-020-01612-y

Unterschied Policy Gradient und Deep Q-Learning

Merkmal	Deep Q-Learning	Policy Gradient
Art des Lernens	Versuch für jeden State Action-Value Function zu finden	Lernt welche Aktion bevorzugt werden sollte
Input	Current State	Current State
Output	Qualität einer Aktion	Wahrscheinlichkeit Aktion zu wählen
Aktionswahl	Bellman Equation	Gewichtete Stichprobe über Aktion
Vorteil	Einfachheit; diskreter Table	Großer, kontinuierlicher Aktionsraum möglich
	Schnelligkeit bei kleinem Aktionsraum	Aktionsraum kann unendlich sein
Gemeinsamkeiten	States, Actions, Rewards, neuronale Netze	

- Verbindung beider Verfahren möglich → Actor Critic

Sources:
Einfache Erläuterung: https://towardsdatascience.com/an-intuitive-explanation-of-policy-gradient-part-1-reinforce-aa4392cbfd3c
Sehr ausführliche Beschreibung: https://towardsdatascience.com/qrash-course-ii-from-q-learning-to-gradient-policy-actor-critic-in-12-minutes-8e8b47129c8c

Paper: Designing an adaptive production control system using reinforcement learning

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN
UNIVERSITY OF APPLIED SCIENCES

1. Action Module

- Output des RL Algorithmus ist diskret → Dieser muss einer der ausführbaren Aktionen zugeordnet werden.

  ~~Zwei Alternativen werden betrachtet:~~

  - ~~Direct mapping: RL-A gibt Action value zurück, der direkt auf Action gemapped wird~~
  - ~~Resource mapping: RL-A gibt diskretes Wertepaar zurück → Quell- und Zielressource, welches damit auszuführende Aktion gibt~~

- ~~Es wird zwischen gültigen und ungültigen Aktionen unterschieden~~
  - ~~Aktion ist bspw. ungültig, wenn Maschine nicht genug Puffer hat~~

Sources:
https://link.springer.com/article/10.1007/s10845-020-01612-y

2. State Module

- Entscheidungsrelevante Informationen werden in einzelnem numerischen **State Vector** verbunden
- Besteht aus:
  - S_{AS}: Binärvariable ob Aktion *valid* oder *invalid* ist $as_i := \begin{cases} 1 & a_i \in A^t_{valid} \\ 0 & \text{else} \end{cases}$
  - S_L: One hot vector für jede Ressource und ob Dispatcher dort ist $l_i := \begin{cases} 1 & \text{if the dispatcher is at resource } i \\ 0 & \text{else} \end{cases}$
  - S_{MF}: Maschinenfehler für jede Maschine $mf_i := \begin{cases} 1 & \text{if } M_i \text{ has a failure} \\ 0 & \text{else} \end{cases}$
  - S_{RPT}: Remaining Process Time von Maschine $rpt_i := \frac{RPT_i}{APT_i}$
  - S_{BEN}: Free buffer space anhand Kapazität $ben_i := 1 - \frac{OCC_i^{EN}}{CAP_i^{EN}}$
  - S_{BPT}: Total processing time of waiting order in front of machine $bpt_i := \frac{\sum_k PT_i^k}{CAP_i^{EN} APT_i} - 1$
  - S_{WT}: Waiting time of orders for transport $wt_i = \frac{WT_i^{max} - WT_i^{mean}}{WT_i^{std}}$
  - S_{AT}: passed time of chosen Action $at_i := \begin{cases} \frac{t_{\to O} + t_{O \to D} + t_{load} + t_{unload}}{at_{max}} & a_i \in A^t_{valid} \\ 0 & \text{else} \end{cases}$

Sources:
https://link.springer.com/article/10.1007/s10845-020-01612-y

State Vector

as_i	as_{i+1}	l_i	l_{i+1}	mf_i	mf_{i+1}	rpt_i	rpt_{i+1}	ben_i	ben_{i+1}	bpt_i	bpt_{i+1}	wt_i	wt_{i+1}	at_i	at_{i+1}

S_{AS} S_L ....

Eher von oben nach unten und key-value Paare (Definition von dict)

Sources:
https://link.springer.com/article/10.1007/s10845-020-01612-y

State Update Code

```
if 'order_waiting_time' in self.parameters['TRANSP_AGENT_STATE']:
    state_type = 'float'
    state = [0.0] * (self.parameters['NUM_MACHINES'] + self.parameters['NUM_SOURCES'])
    for order in Transport.all_transp_orders:
        state[order.current_location.id] += order.get_total_waiting_time()
    result_state.extend(state)
```

```
if 'bin_location' in self.parameters['TRANSP_AGENT_STATE']:
    state_type = 'bool'
    state = [False] * (self.parameters['NUM_MACHINES'] + self.parameters['NUM_SOURCES'] + self.parameters['NUM_SINKS'])
    state[self.current_location.id] = True
    result_state.extend(state)
```

```python
def states(self):
    state_type = 'bool'
    number = 0
    # Avaliable Action are always part of state vector
    if self.parameters['TRANSP_AGENT_ACTION_MAPPING'] == 'direct':
        number += len(self.resources['transps'][0].mapping)
    elif self.parameters['TRANSP_AGENT_ACTION_MAPPING'] == 'resource':
        number += (len(self.resources['transps'][0].mapping) - 1) ** 2 + 1
    # State value alternatives sorted according to the type
    if 'bin_buffer_fill' in self.parameters['TRANSP_AGENT_STATE']:
        number += self.parameters['NUM_MACHINES'] + self.parameters['NUM_SOURCES']
    if 'bin_location' in self.parameters['TRANSP_AGENT_STATE']:
        number += self.parameters['NUM_MACHINES'] + self.parameters['NUM_SOURCES'] + self.parameters['NUM_SINKS']
    if 'bin_machine_failure' in self.parameters['TRANSP_AGENT_STATE']:
        number += self.parameters['NUM_MACHINES']
    if 'int_buffer_fill' in self.parameters['TRANSP_AGENT_STATE']:
        state_type = 'int'
        number += self.parameters['NUM_MACHINES'] + self.parameters['NUM_SOURCES']
    if 'rel_buffer_fill' in self.parameters['TRANSP_AGENT_STATE']:
        state_type = 'float'
        number += self.parameters['NUM_MACHINES'] + self.parameters['NUM_SOURCES']
    if 'rel_buffer_fill_in_out' in self.parameters['TRANSP_AGENT_STATE']:
        state_type = 'float'
        number += self.parameters['NUM_MACHINES'] * 2 + self.parameters['NUM_SOURCES']
    if 'order_waiting_time' in self.parameters['TRANSP_AGENT_STATE']:
        state_type = 'float'
        number += self.parameters['NUM_MACHINES'] + self.parameters['NUM_SOURCES']
    if 'order_waiting_time_normalized' in self.parameters['TRANSP_AGENT_STATE']:
        state_type = 'float'
        number += self.parameters['NUM_MACHINES'] + self.parameters['NUM_SOURCES']
    if 'distance_to_action' in self.parameters['TRANSP_AGENT_STATE']:
        state_type = 'float'
        number += self.parameters['NUM_MACHINES'] + self.parameters['NUM_SOURCES']
    if 'remaining_process_time' in self.parameters['TRANSP_AGENT_STATE']:
        state_type = 'float'
        number += self.parameters['NUM_MACHINES']
    if 'total_process_time' in self.parameters['TRANSP_AGENT_STATE']:
        state_type = 'float'
        number += self.parameters['NUM_MACHINES']
```

- Gespeichert als dict

Link zur Datei:
'AndreasKuhnle/SimRLFab/blob/375b6b0673689d1eb1dd6b19b06643e1112c3fa3/production/envs/production_env.py

3. Reward Module

- RL Agent erhält Rückmeldung in Form eines numerischen *reward signal*
- Gibt zwei verschiedene Varianten
  - 1. Dense reward function → Reward wird nach jedem Schritt gegeben
  - 2. Sparse reward function → Reward wird nach jeder Episode gegeben
    - Agent muss Weg über mehrere Aktionen gehen, um zu wissen welchen Impact diese Aktionen haben

→ Mathematische Erläuterungen erstmal nicht interessant

Sources:
https://link.springer.com/article/10.1007/s10845-020-01612-y

4. Agent Module

- Implementiert verschiedene RL Algorithmen und deren Parameterkonfigurationen
- In PPC ist feste Episodenanzahl schwer sinnvoll festzulegen
  - Deswegen Festlegung einer Episode als **Zeit fester Länge** (z.B. ein Arbeitstag)
  - Oder Festlegung der **Anzahl durchführbarer Aktionen** in einer Episode (wurde in Paper genutzt)

Sources:
https://link.springer.com/article/10.1007/s10845-020-01612-y

Conclusion aus Paper

- Übergebene States haben Einfluss auf die erlernte Performance
- Modellierung eines Reward Signals ist von zentraler Bedeutung, da es Optimierungsziel darstellt
- Ein informationsreicher Statusraum verbessert Leistung, wenn es Bezug zu Zielen gibt

Sources:
https://link.springer.com/article/10.1007/s10845-020-01612-y

Gründe gegen Reinforcement Learning in der SSOP

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Abgestimmt, dass RL in SSOP aufgrund folgender Schwierigkeiten erstmal zurückgestellt wird:

- Geringe Anzahl verfügbarer Beispiele wodurch Implementierung erschwert wird
  - → Mehr Zeit investieren und ggbfs. gemeinsam programmieren
- Reward hängt aufgrund verschiedener Einflüsse der Produktion nicht zwingend vom Zustand während der Aktion ab
  - → Nach hoher Trainingsdauer kann Erfolg erzielt werden
  - → Wenn sich Produktion ändert muss Modell erneut trainiert werden
- RL Frameworks in Python → Fehlende Wrapper machen Integration aufwändig
  - Tests, ob und in wie weit RL zur Problemlösung beiträgt, sind aufwändig
- Zeitlicher Aufwand vs. Ungewissheit über Erfolg

→ **Entwicklung erster Supervised Szenarien in der SSOP mit ML.NET**

- Implementierung aufgrund vorhandener .NET Frameworks erfolgsversprechender

Optimierung des Freigabezeitpunkts

Zu definieren:

- Alle Zustände die Einfluss auf den Liefertermin haben (müssen natürlich nicht alle sein, aber Trainingszeit würde es verkürzen)
- **Actions in NN:** Alle Werte von frühester Startzeitpunkt bis spätester Startzeitpunkt (hier könnte man auch nur ganzzahlige Minutenwerte o.ä. nehmen)

Szenario:

- Agent entscheidet für jeden Auftrag anhand des aktuellen Zustands (Produktionsdauer, aktuelle Auslastung, usw.) wann ein Auftrag freigegeben wird. Feedback erhält er nach Lieferung des Auftrags und Gewichte werden angepasst.

Schwierigkeiten:

- Definierung aller Einflüsse ←→ hohe Trainingsdauer ohne Erfolg
- Beziehung zwischen spätem Reward und ausgewählter Aktion
  - Aktueller Zustand und ausgewählte Aktion hängen durch Einflussfaktoren während der Produktion nicht zwangsläufig vom Ergebnis ab